Java hashCode() Deep Dive

Marcus Biel, Software Craftsman

String hashCode() Example

```
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char val[] = value;

        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
```

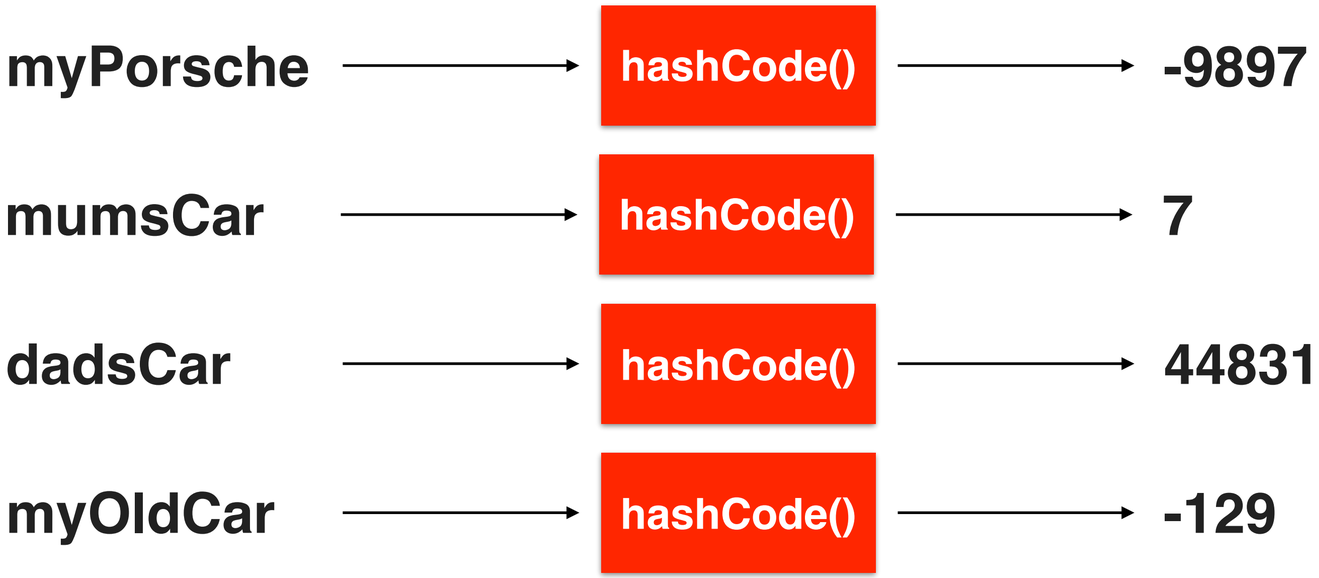
source code from http://openjdk.java.net/

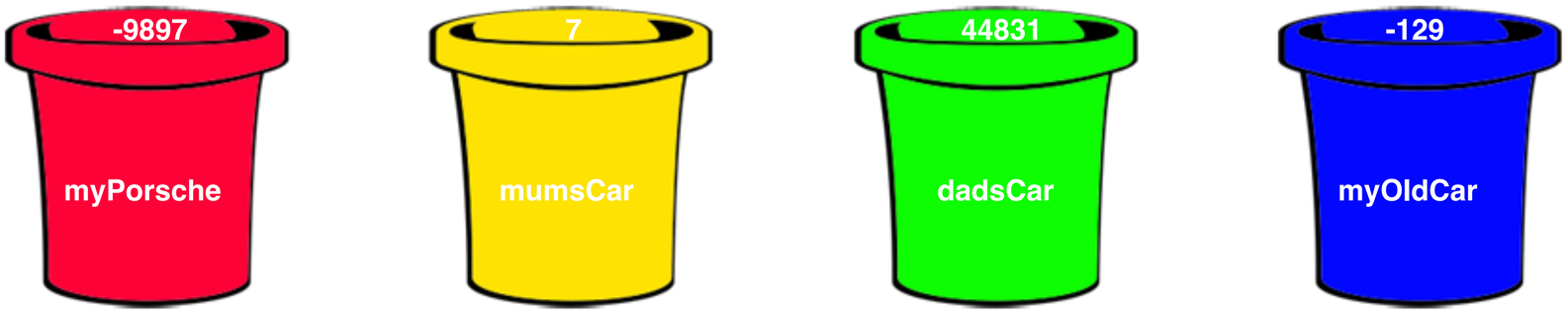
long hashCode() Example

```
public int hashCode() {
  return (int)(value ^ (value >>> 32));
}
```

source code from http://openjdk.java.net/

hashCode()


myPorsche
hashCode()
-9897
mumsCar
hashCode()
7
dadsCar
hashCode()
44831
myOldCar
hashCode()
-129


-9897
myPorsche
7
mumsCar
44831
dadsCar
-129
myOldCar

collisions

collisions

String to hashCode

a naive implementation

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Amy =

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

```
Amy = A
```

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	**13**	14	15	16	17	18	19	20	21	22	23	24	25	26

```
Amy = A + m
```

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	**25**	26

```
Amy = A + m + y
```

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

```
Amy = A + m + y = 1 + 13 + 25 = 39
```

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Amy = A + m + y = 1 + 13 + 25 = 39

May =

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

```
Amy = A + m + y = 1 + 13 + 25 = 39

May = M + a + y =
```

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

```
Amy = A + m + y = 1 + 13 + 25 = 39

May = M + a + y = 13 + 1 + 25 = 39
```

collision

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

```
Amy = A + m + y =  1  +   13 +   25 =
```

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Amy = A + m + y =1*1 + 2*13 + 3*25 =

Mapping

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Amy = A + m + y =1*1 + 2*13 + 3*25 =102

May = M + a + y =1*13 + 2*1 + 3*25 =90

collision free

String hashCode()

java.lang.String hashCode()

Each Character is encoded in Unicode.

Unicode is a mapping table where each character gets assigned to a unique int value

A	M	a	m	y
65	77	97	109	121

java.lang.String hashCode()

A	M	a	m	y
65	77	97	109	121

Amy = ?

int result = 0;

result = 31 * result + 65; //65

result = 31 * result + 109; //4139

result = 31 * result + 121; //65965

result = 65965;

java.lang.String hashCode()

A	M	a	m	y
65	77	97	109	121

May = ?

int result = 0;

result = 31 * result + 77; //77

result = 31 * result + 97; // 2484

result = 31 * result + 121; //77125

result = 77125;

collision free

Why is 31 used?

*“The value 31 was chosen because it is an odd prime. If it were even and the multiplication overflowed, information would be lost, as multiplication by 2 is equivalent to shifting. The advantage of using a prime is less clear, but it is traditional. A nice property of 31 is that the multiplication can be replaced by a shift and a subtraction for better performance: 31 * i == (i << 5) - i. Modern VMs do this sort of optimization automatically.”*

(Josh Bloch, Effective Java, 2nd Edition)

Why is 31 used?

https://computinglife.wordpress.com/2008/11/20/why-do-hash-functions-use-prime-numbers/

“Researchers found that using a prime of 31 gives a better distribution to the keys, and lesser no of collisions. No one knows why, the last i know and i had this question answered by Chris Torek himself, who is generally credited with coming up with 31 hash, on the C++ or C mailing list a while back.”

long hashCode()

long hashCode()

(int) (l ^ (l >>> 32))

(int) (l ^ (l >>> 32))

unsigned bitwise shift (32-bit) >>>

(int) (l ^ (l >>> 32))

exclusive or (xor) ^

(int) (l ^ (l >>> 32))

explicit cast to int

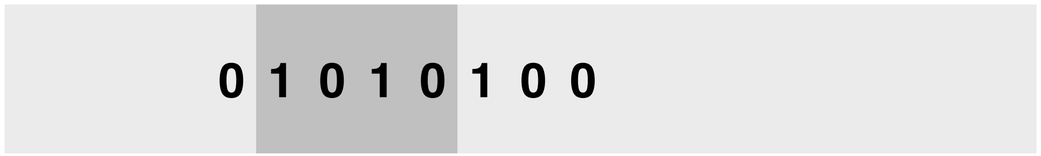
```
long l = 3_050_440_860_235_710_954L;

int i = -1_057_319_386;
```

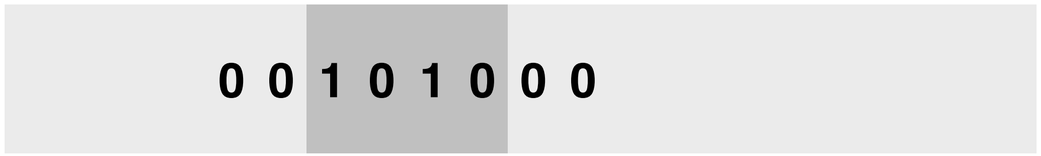
unsigned bitwise shift >>> 4

unsigned **bitwise shift** >>> 4

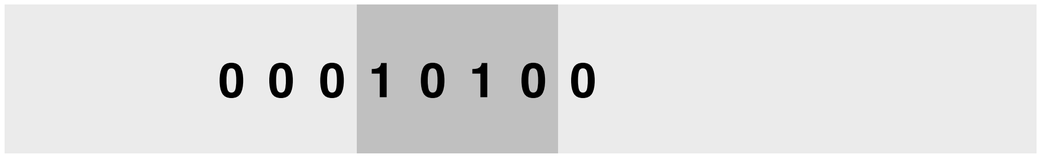
unsigned **bitwise shift** >>> 4



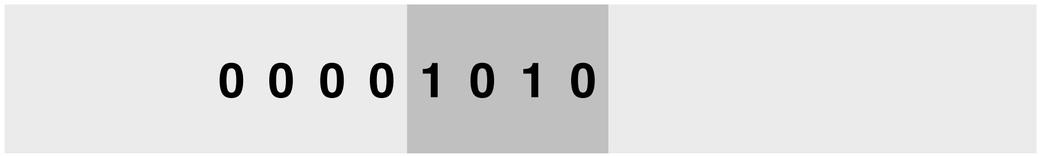
unsigned **bitwise shift** >>> 4



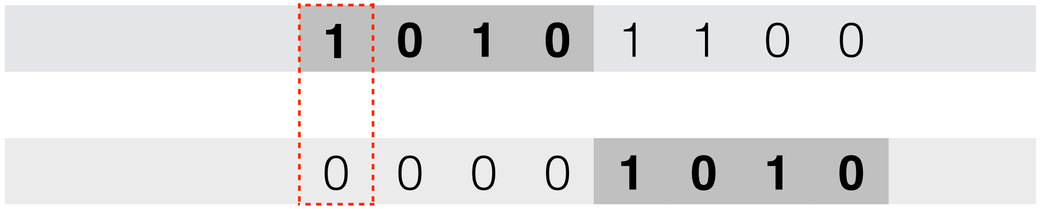
unsigned **bitwise shift** >>> 4



unsigned **bitwise shift** >>> 4



unsigned bitwise shift >>> 4

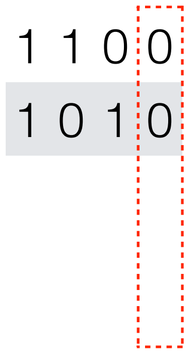


Exclusive or (xor) ^

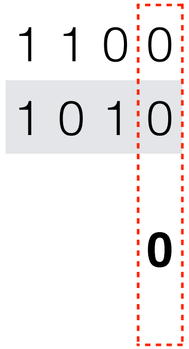
1 1 0 0

1 0 1 0

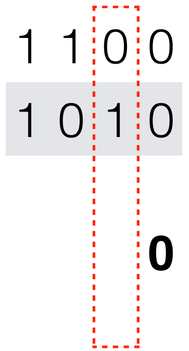
Exclusive or (xor) ^



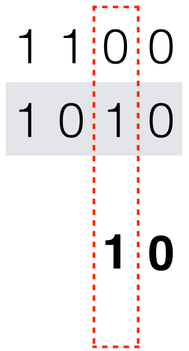
Exclusive or (xor) ^



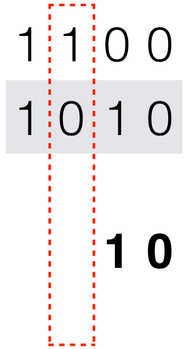
Exclusive or (xor) ^



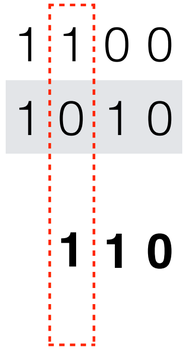
Exclusive or (xor) ^



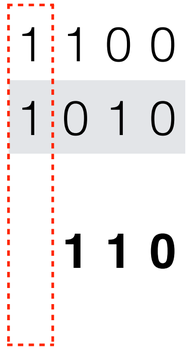
Exclusive or (xor) ^



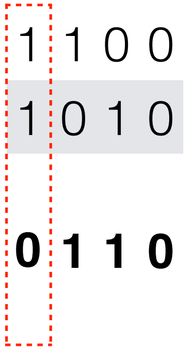
Exclusive or (xor) ^



Exclusive or (xor) ^



Exclusive or (xor) ^



Explicit cast long to int

0010101001010101010101111111001100 11000000111110101001011000100110

Explicit cast long to int

00101010010101010101011111001100 **11000000111110101001011000100110**

Explicit cast long to int

11000000111110101001011000100110

long to int hash value

00101010010101010101011111001100 11101010101011111100000111101010
3,050,440,860,235,710,954 (long)

long to int hash value

```
00101010010101010101011111001100 11101010101011111100000111101010
00101010010101010101011111001100 11101010101011111100000111101010
```

long to int hash value

```
00101010010101010101011111001100 11101010101011111100000111101010
00000000000000000000000000000000 00101010010101010101011111001100
```

long to int hash value

```
00101010010101010101011111001100 11101010101011111100000111101010
00000000000000000000000000000000 00101010010101010101011111001100
00101010010101010101011111001100 11000000111111010100101100010011O
```

long to int hash value

```
0010101001010101010111111001100 11101010101011111100000111101010
0000000000000000000000000000000 00101010010101010101011111001100
                                11000000111111010100101100010011O
```

long to int hash value

11000000111110101001011000100110

```
(int) ( l ^ (l >>> 32))
```

float hashCode()

float hashCode()

return Float.floatToIntBits(f);

double hashCode()

double hashCode()

long l = Double.doubleToLongBits(d);

double hashCode()

```
long l = Double.doubleToLongBits(d);
return Long.hashCode(l);
```

boolean hashCode()

boolean hashCode()

if(b) return 1231;
else return 1237;

byte, char, short int hashCode()

return (int) i;