SAN FRANCISCO STATE UNIVERSITY
COMPUTER SCIENCE DEPARTMENT

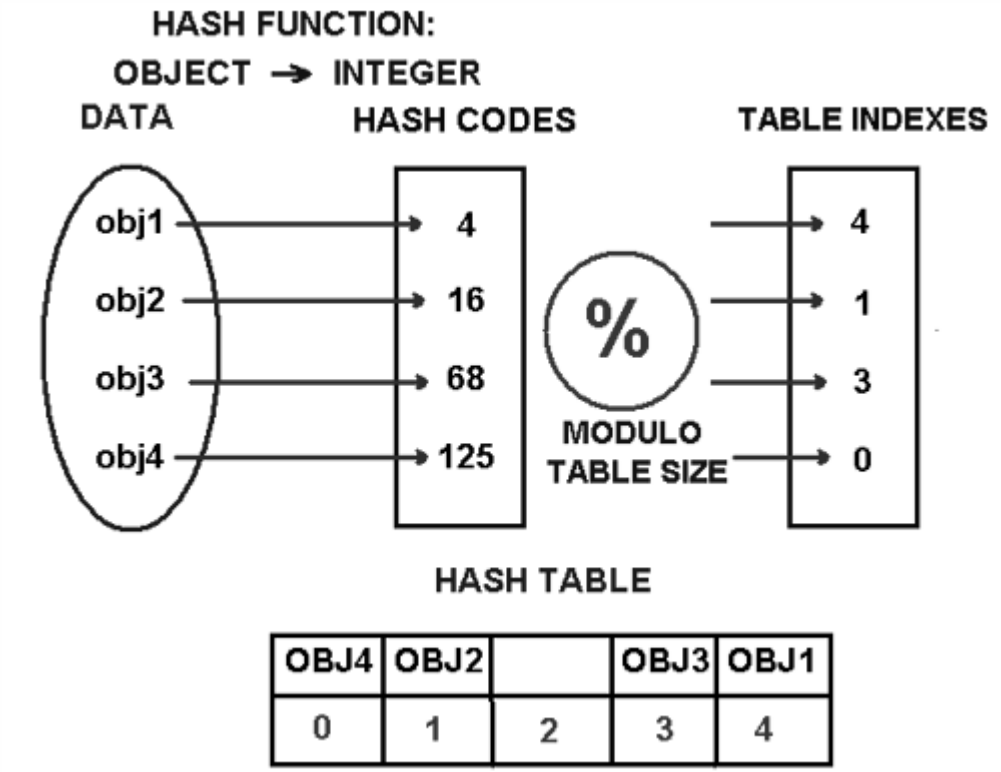
SF STATE

HASHCODE() & HASHING

DUC TA

EXPERIENTIA DOCET

HASHCODE() AND HASHING



HASHCODE() AND HASHING

hashCode

```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (The hashCode may or may not be implemented as some function of an object's memory address at some point in time.)

Returns:

a hash code value for this object.

See Also:

`equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

https://docs.oracle.com/javase/10/docs/api/java/lang/Object.html#hashCode()

Oracle java.util.Objects::hashCode

- Method **hashCode()** method **digests** the data stored in an instance of a class into a single hash value (a 32-bit signed integer).

- Method hashCode() is a native method. It has the modifier “native” and is implemented directly in the native code in the JVM:

```
java.lang.Object::hashCode()

public native int hashCode();
```

- OpenJDK: http://hg.openjdk.java.net/jdk7/jdk7/jdk

HASHCODE() AND HASHING

equals

`public boolean equals(Object obj)`

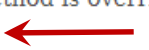
Indicates whether some other object is "equal to" this one.

The `equals` method implements an equivalence relation on non-null object references:

- It is *reflexive*: for any non-null reference value `x`, `x.equals(x)` should return `true`.
- It is *symmetric*: for any non-null reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any non-null reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any non-null reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`, provided no information used in `equals` comparisons on the objects is modified.
- For any non-null reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x == y` has the value `true`).

Note that it is generally necessary to override the `hashCode` method whenever this method is overridden, so as to maintain the general contract for the `hashCode` method, which states that equal objects must have equal hash codes.



Parameters:

`obj` - the reference object with which to compare.

Returns:

`true` if this object is the same as the obj argument; `false` otherwise.

See Also:

`hashCode()`, `HashMap`

hashCode() and equals()

- If two objects are equal, then their hashCode values must be equal.
- If two objects have equal hashCode values, they are not necessarily equal.
- Method hashCode() does not provide unique identifier for an object
- If a class overrides method equals(), it must override method hashCode().
- Hash code computation should not include any field that is not used for equality check (or any field that is not essential to the object). The set of fields used for hashing should be a subset of the fields used for equality.
- If a hash-relevant field changes, the hash code is not recomputed. The internal array is not updated. Use mutable fields when possible.
- A good hashing algorithm produces as few collisions as possible or as few items in a same bucket as possible.

hashMap()

- When an element is added to a hash map, its hash code is used to compute the index in an internal array which is called a bucket.

- If one or more non-equal elements have the same hash code, they are stored in the same bucket. They must be bundled together in a collection such as a list.

- When an instance is given to method **contains(),** its hash code is used to compute the bucket. Only elements therein are compared to the instance.

Thus, this approach reduces the number of potentially equal instances before comparing them.

GOOGLE GUAVA

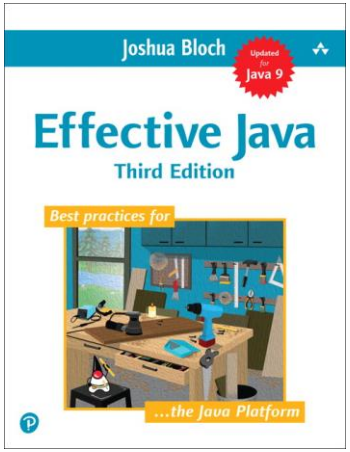
CODE LIBRARIES FOR JAVA AND ANDROID

Guava hashCode()

- **URL:** https://github.com/google/guava/blob/master/guava/src/com/google/common/hash/HashCode.java

JOSHUA BLOCH

EFFECTIVE JAVA



Joshua Bloch's hashCode()

- **Sun Microsystems and Google:** https://en.wikipedia.org/wiki/Joshua_Bloch
- **Oracle and Google:** http://www.oracle.com/technetwork/articles/java/bloch-effective-08-qa-140880.html
- **Effective Java:** https://www.pearson.com/us/higher-education/program/Bloch-Effective-Java-3rd-Edition/PGM1763855.html

Joshua Bloch's hashCode()

- **Sun Microsystems and Google:** https://en.wikipedia.org/wiki/Joshua_Bloch
- **Oracle and Google:** http://www.oracle.com/technetwork/articles/java/bloch-effective-08-qa-140880.html
- **Effective Java:** https://www.pearson.com/us/higher-education/program/Bloch-Effective-Java-3rd-Edition/PGM1763855.html

A short version

1. Create a `int result` and assign a **non-zero** value.
2. For *every field* `f` tested in the `equals()` method, calculate a hash code `c` by:
   - If the field f is a `boolean`: calculate `(f ? 0 : 1)`;
   - If the field f is a `byte`, `char`, `short` or `int`: calculate `(int)f`;
   - If the field f is a `long`: calculate `(int)(f ^ (f >>> 32))`;
   - If the field f is a `float`: calculate `Float.floatToIntBits(f)`;
   - If the field f is a `double`: calculate `Double.doubleToLongBits(f)` and handle the return value like every long value;
   - If the field f is an *object*: Use the result of the `hashCode()` method or 0 if `f == null`;
   - If the field f is an *array*: see every field as separate element and calculate the hash value in a *recursive fashion* and combine the values as described next.
3. Combine the hash value `c` with `result`:

```
result = 37 * result + c
```

4. Return `result`

HASHCODE()

OTHER IMPLEMENTATIONS

More Samples

```
// Simple Implementation

@Override
public int hashCode() {
    return (int) id * fName.hashCode() *
lName.hashCode();
}
```

More Samples

```
// Standard Implementation

@Override
public int hashCode() {
    int hash = 1;
    hash = 37 * hash + (int) id;
    hash = 37 * hash + (fName == null ? 0 : fName.hashCode());
    hash = 37 * hash + (lName == null ? 0 : lName.hashCode());
    return hash;
}
```

More Samples

```
// IntelliJ IDEA

@Override
public int hashCode() {
    int result = (int) (id ^ (id >>> 32));
    result = 37 * result + fName.hashCode();
    result = 37 * result + lName.hashCode();
    return result;
}
```

More Samples

```
// Eclipse

@Override
public int hashCode() {
    final int prime = 37;
    int result = 1;
    result = prime * result + ((fName == null) ? 0 : lName.hashCode());
    result = prime * result + (int) (id ^ (id >>> 32));
    result = prime * result + ((fName == null) ? 0 : lName.hashCode());
    return result;
}
```

See you next class!