The mysterious hash

How can I find out if my delicious hash contains a specific “ingredient” in constant time?

Announcements

• Apt Set 2 - Due tonight

• Jotto – Due Sept 24
Primitives

```java
int i = 5;
int j = 5;

if(i == j)
    doSomething();
else
    doSomethingElse();
```

Objects

```java
int[] array1 = new int[5];
int[] array2 = new int[5];

if(array1 == array2)
    doSomething();
else
    doSomethingElse();
```
int[] array1 = new int[5];
int[] array2 = new int[5];

if(array1 == array2)
doSomething();
else
doSomethingElse();

• Primitives are saved as values
• Objects are saved as reference values – value that points to a location somewhere in memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>@123</td>
<td>i</td>
<td>5</td>
</tr>
<tr>
<td>@124</td>
<td>j</td>
<td>@397</td>
</tr>
<tr>
<td>@397</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Objects

- Primitives are saved as values
- Objects are saved as reference values – value that points to a location somewhere in memory

- `int[] a = {1,2,3};`
- `int[] b = {1,2,3};`
- Does `a == b`?

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>@123</td>
<td>a</td>
<td>@418</td>
</tr>
<tr>
<td>@124</td>
<td>b</td>
<td>@397</td>
</tr>
</tbody>
</table>

DO NOT USE `==` FOR OBJECTS!
• WARNING

String s1 = “hello”;
String s2 = “hello”;

Eclipse
if (s1 == s2)

APT tester
if (s1 == s2)

MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?

0x3A28213A
0x6339392C
0x7363682E.

I HATE YOU.
Objects

int[] array1 = new int[5];
in[] array2 = new int[5];

if(array1 == array2)
dosomething();
else
dosomethingelse();

How to compare objects?

• Is this the same teapot?

• teapot1.equals(teapot2);
.equals()

- Built in Java function for Object
- All objects inherit .equals()

```java
Circle[] c = new Circle[numCircles];
for (Circle[] c, i++){
    System.out.println("equal");
}
else
    System.out.println("not equal");
```

```java
ThreeInts a = new ThreeInts(5,5,5);
ThreeInts b = new ThreeInts(5,5,5);
if(a.equals(b))
    System.out.println("equal");
else
    System.out.println("not equal");
```
.equals()

- Built in Java function for Object
- All objects inherit .equals()
  - You can Override .equals() with your own code!

```java
Circle[] c = new Circle[numCircles];
for (int i = 0; i < numCircles; i++) {
    c[i] =颜色[i % colors.length];
}

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

- Built in Java function for Object
- All objects inherit .hashCode()
  - You can Override .hashCode() with your own code!

```java
Circle[] c = new Circle[numCircles];
for (Circle circle : circles; i++){
    c[i] = new Circle(i, i, colors[i%numColors]);
}
```

**.hashCode()**

- Hash Yourself!

```java
String name = "Tabitha";
System.out.println(name.hashCode());
```

111673433
Hashing

- HashTable
  - array of fixed size
  - with a key to each location
  - each key is mapped to an index in the table

- Hash function
  - simple to compute
  - ensure two distinct keys get different cells
Hashing

• Hash function
  • simple to compute
  • ensure two distinct keys get different cells

9/15/13

Hashing

• Two equal objects should hash to the same place (have the same key)

9/15/13
• Two equal objects should hash to the same place (have the same key)

```java
if a.equals(b)
    a.hashCode() == b.hashCode()
```

• Hash function
  • simple to compute
  • ensure two distinct keys get different cells
Hashing

- Hash function
  - simple to compute
  - ensure two distinct keys get different cells

- Separate Chaining
  - make your table into a list!
if a.equals(b)
then a.hashCode() == b.hashCode()

HOWEVER

if a.hashCode() == b.hashCode()
then a.equals(b) || !a.equals(b)

Today

• Do NOT use == for objects
• .equals()
• .hashCode()
• HashTables