

## Underflow, cont'd

- So we had this underflow issue...

$$\Pr[C_S | \mathbf{X}] = \Pr[C_S] \prod_{j=1}^k \Pr[x_j | C_S]$$

- Product of many factors in  $[0,1]$  quickly goes to 0...
- Answer: rescale to a more coherent range
- Suggestion 1:
  - For factor  $j$ , if  $\Pr[x_j | C_S] > \Pr[x_j | C_N]$ , then add in 1; else add in 0.
  - Good suggestion, but it's only an approximation.
- Other suggestions?

# Rescaling

- Look again at our posterior prob. eqn:

$$\Pr[C_S|\mathbf{X}] = \Pr[C_S] \prod_{j=1}^k \Pr[x_j|C_S]$$

- Now, look what happens when you take the log:

$$L[C_S|\mathbf{X}] = \log(\Pr[C_S]) + \sum_{j=1}^k \log(\Pr[x_j|C_S])$$

- Now we have a sum of a bunch of terms in the range  $[-\infty, 0]$ . These turn out to be much more reasonably scaled.
- $L[C_S|\mathbf{X}]$  is called the (posterior) *log likelihood*
- That avoids underflow. But... Why should it work? This has changed all the numbers, right?

## More pesky zeros...

- Practical problem #2:
- What if  $\Pr[x_j | C_i] = 0$ ? (I.e., you see a word/ngram in your UNLABELED email that you never saw during TRAINING)
- What happens to  $\Pr[C_i | \mathbf{X}]$ ? What about  $L(C_i | \mathbf{X})$ ?

## Laplace smoothing

- So unknown data kicks butt on our probability (likelihood) estimates
- There's a cheap hack way to fix that...
- Replace your token estimates

$$\Pr[x_j | C_i] = (\# \text{ tok } x_j \text{ in } C_i) / (\# \text{ all toks in } C_i)$$

- With

$$\Pr[x_j | C_i] = (1 + \# \text{ tok } x_j \text{ in } C_i) / (1 + \# \text{ all toks in } C_i)$$

- Now guaranteed never to have a 0 prob.
- This is called a Laplace smoothed estimate
- It adds a bias, but in the limit the bias goes away.

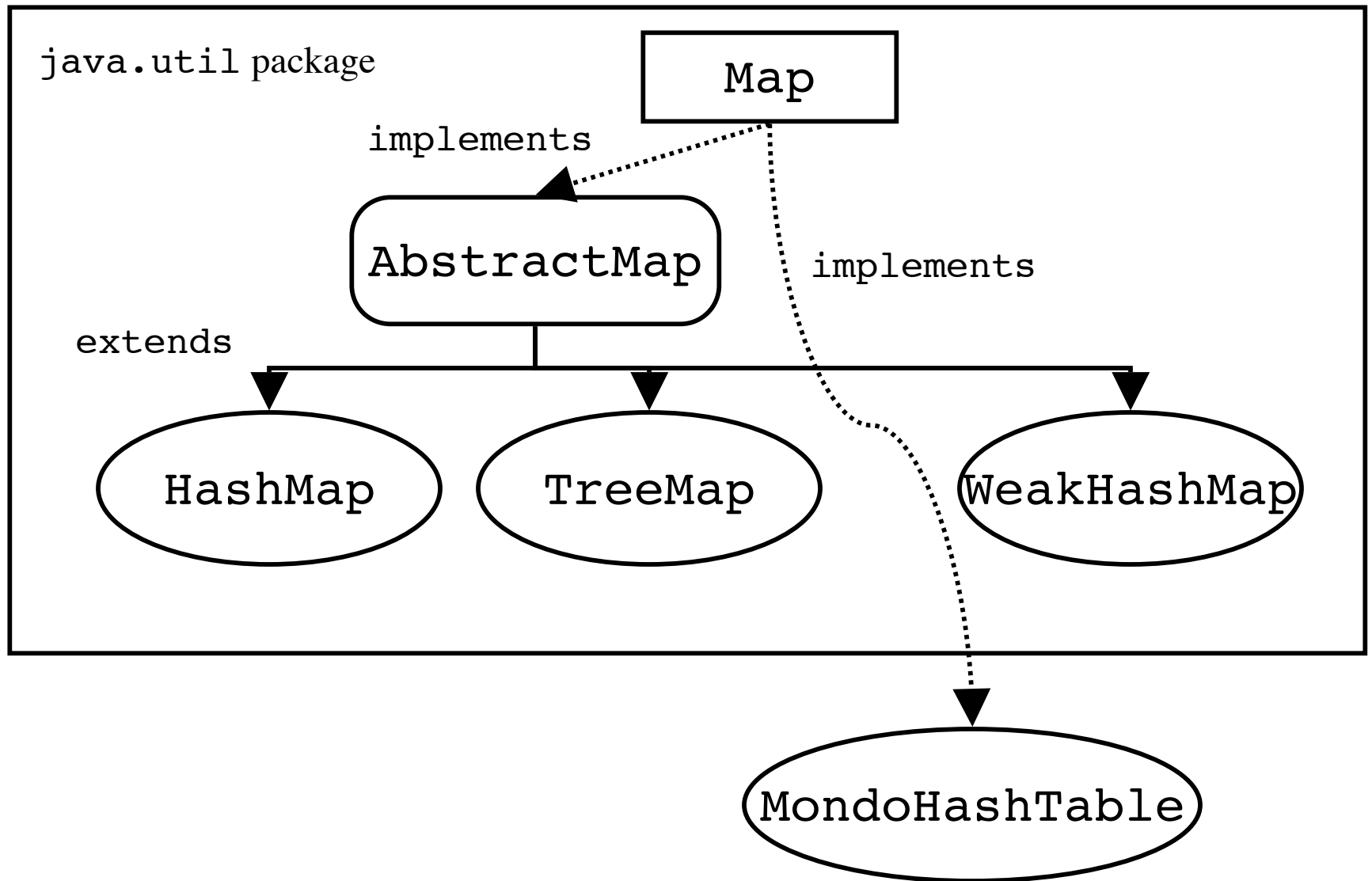
# Interfaces and all that

---

# What's an interface, anyway?

- An interface...
  - Gives definitions for all of the *methods* that a class must support
  - Does *not* provide any code for any of those methods
  - Does *not* provide any fields (i.e., data)
- Essentially, an interface is a contract or specification for *how a class needs to work*
- It does not say *anything* about *how* the class is to get that job done...

# The java.util.Map interface



## Two other types of classes...

- The `AbstractMap` class: an *abstract class*
  - Provides actual code for (most) methods of an interface
  - Probably provides some private/protected data (I.e., fields)
  - Does *not* provide *every* method
  - Can't actually make (`new`) one of these things...
- Exists for convenience -- makes it *much* easier to make an actual `Map` class
- (This is why you can't use `AbstractMap` in your project)

## Concrete classes

---

- E.g., HashMap or TreeMap
- Provide *complete* code and data to make the entire class work
- These are things you can actually instantiate (I.e., create with `new`)
- Your `MondoHashTable` will be a concrete class (though you can introduce your own abstract intermediate class if you feel it necessary)