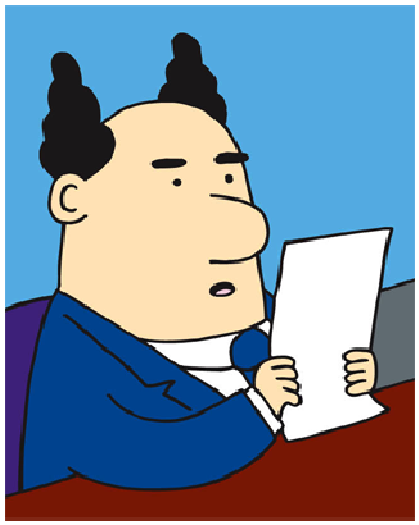
Getting Fancy w/ Generic Types

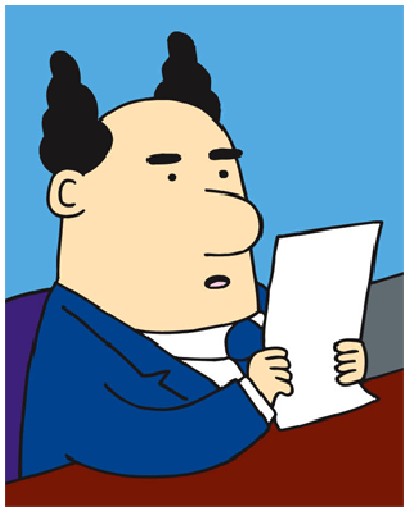
…



New Requirement!

We need more than one Type Parameter

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}
```



New Requirement!

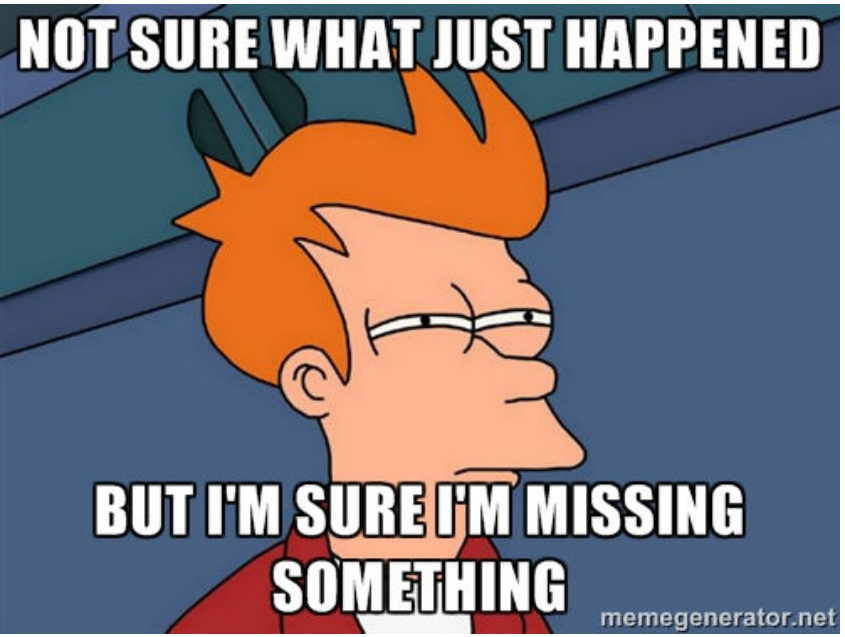
We need more than one Type Parameter

```
public class OrderedPair<K, V> implements Pair<K, V>
{
    private K key;
    private V value;

    public OrderedPair(K key, V value){
        this.key = key;
        this.value = value;
    }

    public K getKey()       {  return key;   }
    public V getValue()     {  return value; }
}
```

Example of using Generic Types w/ Multiple Type Parameters

```
Pair<String, Integer> p1;
p1 = new OrderedPair<String, Integer>("Even", 8);

Pair<String, String>  p2;
p2 = new OrderedPair<String, String>("hello", "world");
```

What is wrong with this?



Let us look at the constructor's arguments…

Actual Type Parameter = String

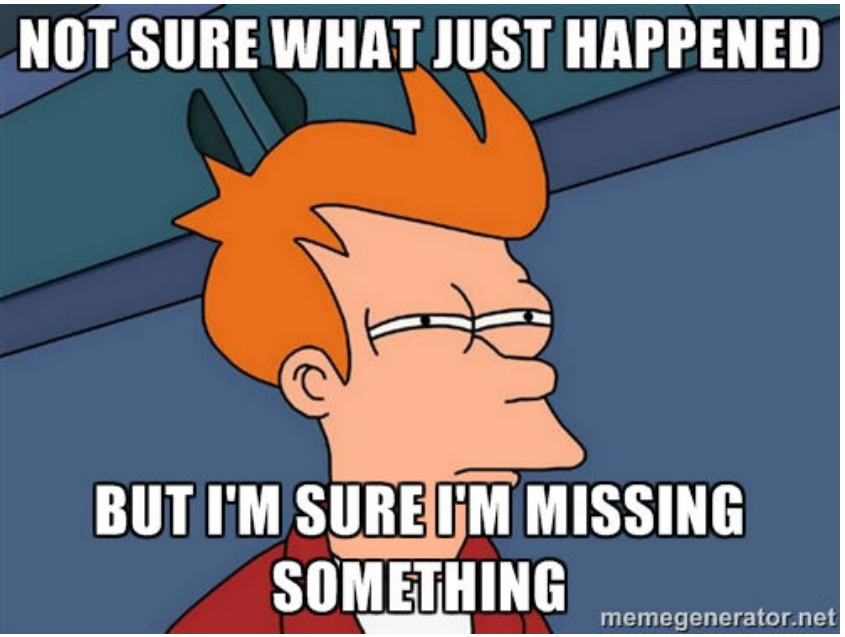
Actual Type Parameter = Integer

```
Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
```

So why is it working?

✓ Actual Constructor Parameter = String

✗ Actual Constructor Parameter = int



Let us look at the constructor's arguments…

```
Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
```

Integer.valueOf(8)

? So why is it working?

AutoBoxing

Last but not Least…

A Type Parameter may be…

…a Parameterized Type!

```
OrederedPair<String,                    > p;

p =    new OrderedPair<>(
            "primes",
                                        );
```