Representing Emptiness In Scala (with Null, null, Nothing, Nil, None, and Unit)

There are a [couple](http://lousycoder.com/index.php?/archives/92-Scala-So-many-ways-to-say-nothing.html) of [explanations](http://oldfashionedsoftware.com/2008/08/20/a-post-about-nothing/) of these terms out there already, but my needs weren’t quite satisfied by them. Detail is great, but I wanted concise high level descriptions that explain what the primary purpose for each construct is and emphasize the ones you’re most likely to see in practice. This explanation is intended to *complement* ones with more detail, not replace them.

**null**

Scala’s null is the same as in Java. Any reference type can be null, like Strings, Objects, or your own classes. Also just like Java, value types like Ints can’t be null. **Odds are you'll see this one.**

**Null**

[Null](http://www.scala-lang.org/api/current/index.html#scala.Null) is a trait whose only instance is null. It is a subtype of all reference types, but not of value types. It purpose in existing is to make it so reference types can be assigned null and value types can’t. **Don't worry about this one.**

**Nothing**

[Nothing](http://www.scala-lang.org/api/current/index.html#scala.Nothing) is a trait that is guaranteed to have *zero* instances. It is a subtype of all other types. It has two main reasons for existing: to provide a return type for methods that **never** return normally (i.e. a method that always throws an exception). The other reason is to provide a type for Nil (explained below). **Don't worry about this one.**

**Nil**

[Nil](http://www.scala-lang.org/api/current/index.html#scala.collection.immutable.Nil$) is just an empty list, exactly like the result of List(). It is of type List[Nothing]. And since we know there are no instances of Nothing, we now have a list that is statically verifiable as empty. Nice to have. **Odds are you'll see this one.**

**None**

[None](http://www.scala-lang.org/api/current/index.html#scala.None$) is the counterpart to [Some](http://www.scala-lang.org/api/current/index.html#scala.Some), used when you’re using Scala’s [Option](http://www.scala-lang.org/api/current/index.html#scala.Option) class to help avoid null references. If you’re not familiar with the idea of Option or Maybe, here’s an [introduction to Option](http://blog.danielwellman.com/2008/03/using-scalas-op.html). **Odds are you'll see this one.**

**Unit**

[Unit](http://www.scala-lang.org/api/current/index.html#scala.Unit) in Scala is the equivalent of void in Java. It’s used in a function’s signature when that function doesn’t return a value. **Odds are you'll see this one.**

Scala FAQ: What is the difference between Nil and List() in [Scala](http://scala-lang.org/)?

Short answer: There isn’t any difference, as shown in the Scala REPL:

scala> **Nil == List()**

res0: Boolean = true

It’s more “idiomatic Scala” Scala to use Nil rather than List(). For instance, I wrote code like this last night using Nil in a Scala match/case expression:

post match {

case '\n' :: rest => splitString(post, acc += pre.mkString)

case Nil => acc += pre.mkString

case \_ => Nil

}

But either Nil or List() will work in that situation.

## Java’s System.identityHashCode

As a longer answer, [this SO post](http://stackoverflow.com/questions/5981850/scala-nil-vs-list) shows a method named System.identityHashCode that I didn’t know about:

scala> **System.identityHashCode(Nil)**

374527572

scala> **System.identityHashCode(List())**

374527572

As I’ll demonstrate shortly, this shows that Nil and List() have the same hash code value. [Oracle’s System Javadoc](http://docs.oracle.com/javase/7/docs/api/java/lang/System.html) states this about identityHashCode:

"Returns the same hash code for the given object as would be returned by the default method hashCode(), whether or not the given object's class overrides hashCode()."

That’s a nice utility.

## Manually comparing hash codes

Having seen that fancy approach, I now realize that I can compare Nil and List() in the Scala REPL like this:

scala> **Nil.hashCode**

res1: Int = 473519988

scala> **List().hashCode**

res2: Int = 473519988