# Enumerations



An enumeration is used to define a type consisting of a set of named values.

```
enum Color:
  case Red, Green, Blue
```

This defines a new `sealed` class, `Color`, with three values, `Color.Red`, `Color.Green`, `Color.Blue`. The color values are members of `Color`s companion object.

## Parameterized enums

Enums can be parameterized.

```
enum Color(val rgb: Int):
  case Red   extends Color(0xFF0000)
  case Green extends Color(0x00FF00)
  case Blue  extends Color(0x0000FF)
```

As the example shows, you can define the parameter value by using an explicit extends clause.

## Methods defined for enums

The values of an enum correspond to unique integers. The integer associated with an enum value is returned by its `ordinal` method:

```
scala> val red = Color.Red
val red: Color = Red
scala> red.ordinal
val res0: Int = 0
```

The companion object of an enum also defines three utility methods. The `valueOf` method obtains an enum value by its name. The `values` method returns all enum values defined in an enumeration in an `Array`. The `fromOrdinal` method obtains an enum value from its ordinal ( `Int` ) value.

```
scala> Color.valueOf("Blue")
val res0: Color = Blue
scala> Color.values
val res1: Array[Color] = Array(Red, Green, Blue)
scala> Color.fromOrdinal(0)
val res2: Color = Red
```

# User-defined members of enums

It is possible to add your own definitions to an enum. Example:

```
enum Planet(mass: Double, radius: Double):
  private final val G = 6.67300E-11
  def surfaceGravity = G * mass / (radius * radius)
  def surfaceWeight(otherMass: Double) = otherMass * surfaceGravity

  case Mercury extends Planet(3.303e+23, 2.4397e6)
  case Venus   extends Planet(4.869e+24, 6.0518e6)
  case Earth   extends Planet(5.976e+24, 6.37814e6)
  case Mars    extends Planet(6.421e+23, 3.3972e6)
  case Jupiter extends Planet(1.9e+27,   7.1492e7)
  case Saturn  extends Planet(5.688e+26, 6.0268e7)
  case Uranus  extends Planet(8.686e+25, 2.5559e7)
  case Neptune extends Planet(1.024e+26, 2.4746e7)
end Planet
```

# User-defined companion object of enums

It is also possible to define an explicit companion object for an enum:

```
object Planet:
  def main(args: Array[String]) =
    val earthWeight = args(0).toDouble
    val mass = earthWeight / Earth.surfaceGravity
    for p <- values do
      println(s"Your weight on $p is ${p.surfaceWeight(mass)}")
end Planet
```

# Restrictions on Enum Cases

Enum case declarations are similar to secondary constructors: they are scoped outside of the enum template, despite being declared within it. This means that enum case declarations cannot access inner members of the enum class.

Similarly, enum case declarations may not directly reference members of the enum's companion object, even if they are imported (directly, or by renaming). For example:

```
import Planet.*
enum Planet(mass: Double, radius: Double):
  private final val (mercuryMass, mercuryRadius) = (3.303e+23, 2.4397e6)

  case Mercury extends Planet(mercuryMass, mercuryRadius)             // Not fo
  case Venus   extends Planet(venusMass, venusRadius)                 //
illegal reference
  case Earth   extends Planet(Planet.earthMass, Planet.earthRadius)   // ok
object Planet:
  private final val (venusMass, venusRadius) = (4.869e+24, 6.0518e6)
  private final val (earthMass, earthRadius) = (5.976e+24, 6.37814e6)
end Planet
```

The fields referenced by `Mercury` are not visible, and the fields referenced by `Venus` may not be referenced directly (using `import Planet.*`). You must use an indirect reference, such as demonstrated with `Earth`.

# Deprecation of Enum Cases

As a library author, you may want to signal that an enum case is no longer intended for use. However you could still want to gracefully handle the removal of a case from your public API, such as special casing deprecated cases.

To illustrate, say that the `Planet` enum originally had an additional case:

```
 enum Planet(mass: Double, radius: Double):
   ...
   case Neptune extends Planet(1.024e+26, 2.4746e7)
+  case Pluto   extends Planet(1.309e+22, 1.1883e3)
 end Planet
```

We now want to deprecate the `Pluto` case. First we add the `scala.deprecated` annotation to `Pluto`:

```
 enum Planet(mass: Double, radius: Double):
   ...
   case Neptune extends Planet(1.024e+26, 2.4746e7)
-  case Pluto   extends Planet(1.309e+22, 1.1883e3)
```

```
+
+  @deprecated("refer to IAU definition of planet")
+  case Pluto extends Planet(1.309e+22, 1.1883e3)
 end Planet
```

Outside the lexical scopes of `enum Planet` or `object Planet`, references to `Planet.Pluto` will produce a deprecation warning, but within those scopes we can still reference it to implement introspection over the deprecated cases:

```
trait Deprecations[T <: reflect.Enum] {
  extension (t: T) def isDeprecatedCase: Boolean
}

object Planet {
  given Deprecations[Planet] with {
    extension (p: Planet)
      def isDeprecatedCase = p == Pluto
  }
}
```

We could imagine that a library may use type class derivation to automatically provide an instance for `Deprecations`.

## Compatibility with Java Enums

If you want to use the Scala-defined enums as Java enums, you can do so by extending the class `java.lang.Enum`, which is imported by default, as follows:

```
enum Color extends Enum[Color] { case Red, Green, Blue }
```

The type parameter comes from the Java enum definition and should be the same as the type of the enum. There is no need to provide constructor arguments (as defined in the Java API docs) to `java.lang.Enum` when extending it – the compiler will generate them automatically.

After defining `Color` like that, you can use it like you would a Java enum:

```
scala> Color.Red.compareTo(Color.Green)
val res15: Int = -1
```

For a more in-depth example of using Scala 3 enums from Java, see this test. In the test, the enums are defined in the `MainScala.scala` file and used from a Java source, `Test.java`.

# Implementation

Enums are represented as `sealed` classes that extend the `scala.reflect.Enum` trait. This trait defines a single public method, `ordinal`:

```
package scala.reflect

/** A base trait of all Scala enum definitions */
transparent trait Enum extends Any, Product, Serializable:

  /** A number uniquely identifying a case of an enum */
  def ordinal: Int
```

Enum values with `extends` clauses get expanded to anonymous class instances. For instance, the `Venus` value above would be defined like this:

```
val Venus: Planet = new Planet(4.869E24, 6051800.0):
  def ordinal: Int = 1
  override def productPrefix: String = "Venus"
  override def toString: String = "Venus"
```

Enum values without `extends` clauses all share a single implementation that can be instantiated using a private method that takes a tag and a name as arguments. For instance, the first definition of value `Color.Red` above would expand to:

```
val Red: Color = $new(0, "Red")
```

# Reference

For more information, see Issue #1970 and PR #4003.

‹ Enums Algebr... ›

Contributors to this page

 pikinier20  anatoliykmetyuk  BarkingBad  julienrf

 odersky  ShapelessCat  bishabosha  michelou  mbloms

 fgoret  som-snytt