Main Methods

Scala 3 offers a new way to define programs that can be invoked from the command line: A `@main` annotation on a method turns this method into an executable program. Example:

```
@main def happyBirthday(age: Int, name: String, others: String*) =
  val suffix =
    age % 100 match
    case 11 | 12 | 13 => "th"
    case _ =>
      age % 10 match
        case 1 => "st"
        case 2 => "nd"
        case 3 => "rd"
        case _ => "th"
  val bldr = new StringBuilder(s"Happy $age$suffix birthday, $name")
  for other <- others do bldr.append(" and ").append(other)
  bldr.toString
```

This would generate a main program `happyBirthday` that could be called like this

```
> scala happyBirthday 23 Lisa Peter
Happy 23rd birthday, Lisa and Peter
```

A `@main` annotated method can be written either at the top-level or in a statically accessible object. The name of the program is in each case the name of the method, without any object prefixes. The `@main` method can have an arbitrary number of parameters. For each parameter type there must be an instance of the `scala.util.CommandLineParser.FromString[T]` type class that is used to convert an argument string to the required parameter type `T`. The parameter list of a main method can end in a repeated parameter that then takes all remaining arguments given on the command line.

The program implemented from a `@main` method checks that there are enough arguments on the command line to fill in all parameters, and that argument strings are convertible to the required types. If a check fails, the program is terminated with an error message.

Examples:

```
> scala happyBirthday 22
Illegal command line after first argument: more arguments expected

> scala happyBirthday sixty Fred
Illegal command line: java.lang.NumberFormatException: For input string: "sixty
```

The Scala compiler generates a program from a `@main` method `f` as follows:

- It creates a class named `f` in the package where the `@main` method was found
- The class has a static method `main` with the usual signature. It takes an `Array[String]` as argument and returns `Unit`.
- The generated `main` method calls method `f` with arguments converted using methods in the `scala.util.CommandLineParser` object.

For instance, the `happyBirthDay` method above would generate additional code equivalent to the following class:

```
final class happyBirthday:
  import scala.util.CommandLineParser as CLP
  <static> def main(args: Array[String]): Unit =
    try
      happyBirthday(
        CLP.parseArgument[Int](args, 0),
        CLP.parseArgument[String](args, 1),
        CLP.parseRemainingArguments[String](args, 2))
    catch
      case error: CLP.ParseError => CLP.showError(error)
```

Note: The `<static>` modifier above expresses that the `main` method is generated as a static method of class `happyBirthDay`. It is not available for user programs in Scala. Regular "static" members are generated in Scala using objects instead.

`@main` methods are the recommended scheme to generate programs that can be invoked from the command line in Scala 3. They replace the previous scheme to write program as objects with a special `App` parent class. In Scala 2, `happyBirthday` could be written also like this:

```
object happyBirthday extends App:
  // needs by-hand parsing of arguments vector
  ...
```

The previous functionality of `App`, which relied on the "magic" `DelayedInit` trait, is no longer available. `App` still exists in limited form for now, but it does not support command line arguments and will be deprecated in the future. If programs need to cross-build between Scala 2 and Scala 3, it is recommended to use an explicit `main` method with an `Array[String]` argument instead.

‹ Lazy V... Escape... ›