```scala
package polynomial

import tinyscalautils.lang.InterruptibleConstructor

import scala.annotation.tailrec

object ListPoly:
   def apply(coef1: Int, coefs: Int*): ListPoly =
      require(coef1 == 0 implies coefs.isEmpty)
      new ListPoly(coef1 :: coefs.toList)
end ListPoly

@tailrec
private def trimLeft(cs: List[Int]): List[Int] = cs match
  case Nil        => List(0)
  case 0 :: rest  => trimLeft(rest)
  case _          => cs

final case class ListPoly private (coefs: List[Int]) extends InterruptibleConstructor:

   def eval(x: Double): Double      =
     coefs match
       case Nil => 0.0
       case h :: t =>

         @tailrec
         def loop(acc: Double, rest: List[Int]): Double = rest match
           case Nil => acc
           case k :: ks => loop(acc * x + k, ks)

         loop(h.toDouble, t)

   def + (that: ListPoly): ListPoly =
     def padLeft(cs: List[Int], k: Int): List[Int] =
       if k <= 0 then cs else 0 :: padLeft(cs, k - 1)
```

```
36
37      def add(a: List[Int], b: List[Int]): List[Int] = (a, b) match
38        case (Nil, Nil) => Nil
39        case (ha :: ta, hb :: tb) => (ha + hb) :: add(ta, tb)
40        case _ => throw new IllegalStateException("length mismatch") // shouldn't happen
41
42      val a = this.coefs
43      val b = that.coefs
44      val la = a.length
45      val lb = b.length
46
47      val (aa, bb) =
48        if la < lb then (padLeft(a, lb - la), b)
49        else (a, padLeft(b, la - lb))
50
51      ListPoly(trimLeft(add(aa, bb)))
52
53    def * (n: Int): ListPoly         =
54      if n == 0 then ListPoly(0)
55      else
56        def mul(cs: List[Int]): List[Int] = cs match
57          case Nil => Nil
58          case h :: t => (h * n) :: mul(t)
59
60        ListPoly(mul(coefs))
61
62    def * (that: ListPoly): ListPoly =
63      def shiftRight(cs: List[Int], n: Int): List[Int] =
64        if n <= 0 then cs else shiftRight(cs :+ 0, n - 1)
65
66      def mul(cs: List[Int], deg: Int): ListPoly = cs match
67        case Nil =>
68          ListPoly(0)
```

```
69
70        case a :: rest =>
71          // a * x^deg * that
72          val term =
73            ListPoly(shiftRight(that.coefs.map(_ * a),
   deg))
74
75          term + mul(rest, deg - 1)
76
77      if this.coefs == List(0) || that.coefs == List(0
   ) then
78        ListPoly(0)
79      else
80        mul(this.coefs, this.coefs.length - 1)
81
82    def degree: Int                  = if coefs == List
   (0) then 0 else coefs.length - 1
83    def derivative: ListPoly         =
84      if coefs.length <= 1 then
85        ListPoly(0)
86      else
87        val n = coefs.length - 1
88
89        def loop(cs: List[Int], pow: Int): List[Int] =
   cs match
90          case _ :: Nil => Nil // drop constant term
91          case h :: t => (h * pow) :: loop(t, pow - 1)
92          case Nil => Nil
93
94        ListPoly(loop(coefs, n))
95
96 end ListPoly
97
```