Day 1: Linearization

Example 1: Pure linearization

```scala
class B { print("B ") }


trait U1 { print("U1 ") }
trait U2 { print("U2 ") }
trait V1 { print("V1 ") }
trait V2 { print("V2 ") }

trait T1 extends U1 with V1 { print("T1 ") }
trait T2 extends U2 with V2 { print("T2 ") }
trait T3 extends U2 with V1 { print("T3 ") }

class A extends B with T1 with T2 with T3
```

Rules of linearization

Given the above, the steps for determining linearization for any instance of `A` are:

1. Reverse all but first class: `A T3 T2 T1 B`
2. Replace each trait with its linearization (denoted here as `l(trait)`. Make sure you do step 1 for each sub-linearization!
    - `A T3 T2 T1 B`
    - `A l(T3) l(T2) l(T1) l(B)`
    - `A T3 V1 U2 T2 V2 U2 T1 V1 U1 B`
3. De-dupe from left to right, where the right-most instance wins:
    - `A T3 T2 V2 U2 T1 V1 U1 B`
4. Add top-level classes: `A T3 T2 V2 U2 T1 V1 U1 B AnyRef Any`

Initialization order: right to left of linearization order!

So based off this, we expect the `print` statement to read: `B U1 V1 T1 U2 V2 T2 T3`, where `B` is the top-level parent class and is thus initialized / printed first.

Example 2: Working with kinds of `super` calls

```scala
class MultiplierIdentity {
  def identity: Int = 1
}

trait DoubledMultiplierIdentity extends MultiplierIdentity {
  override def identity: Int = 2 * super.identity
}

trait TripledMultiplierIdentity extends MultiplierIdentity {
  override def identity: Int = 3 * super.identity
}

// all of these are first doubled, then tripled
class ModifiedIdentity1 extends DoubledMultiplierIdentity with TripledMultiplierIdentity

class ModifiedIdentity2 extends DoubledMultiplierIdentity with TripledMultiplierIdentity {
  override def identity: Int = super[DoubledMultiplierIdentity].identity
}

class ModifiedIdentity3 extends DoubledMultiplierIdentity with TripledMultiplierIdentity {
  override def identity: Int = super[TripledMultiplierIdentity].identity
}


object ModifiedIdentityUser {

  def main(args: Array[String]): Unit = {
    val instance1 = new ModifiedIdentity1
    val instance2 = new ModifiedIdentity2
    val instance3 = new ModifiedIdentity3

    println(s"Result 1: ${instance1.identity}") // 6
    println(s"Result 2: ${instance2.identity}") // 2
    println(s"Result 3: ${instance3.identity}") // 6
  }
}
```

Example 3: Lazy vals

```scala
trait base { val a: Int; lazy val b: Int = a }
trait four extends base { override val a: Int = 4 }
trait three extends base { override val a: Int = 3 }
trait two extends base { override val a: Int = 2 }

object one extends four with three with two
println(one.b) // prints 2
```