Adventures in Meta-Programming

Macros vs Shapeless

Dave Gurnell, @davegurnell

meta-programming

[**mee**-t<u>uh</u>-**proh**-gram-ing]

noun

1. the practice of writing code that writes code;

domain specific languages

scrapping boilerplate

meta-programming

[**mee**-t<u>uh</u>-**proh**-gram-ing]

noun

1. the practice of writing code that writes code;
2. time consuming;
3. difficult to get right.

which approach to choose?

how to integrate it?

macros

shapeless

tips for getting started

cool examples!

github.com/davegurnell/macros-vs-shapeless

constructing values

```
case class IceCream(
  name: String,
  numCherries: Int,
  inCone: Boolean)
```

```
create[IceCream]
// IceCream("", 0, false)
```

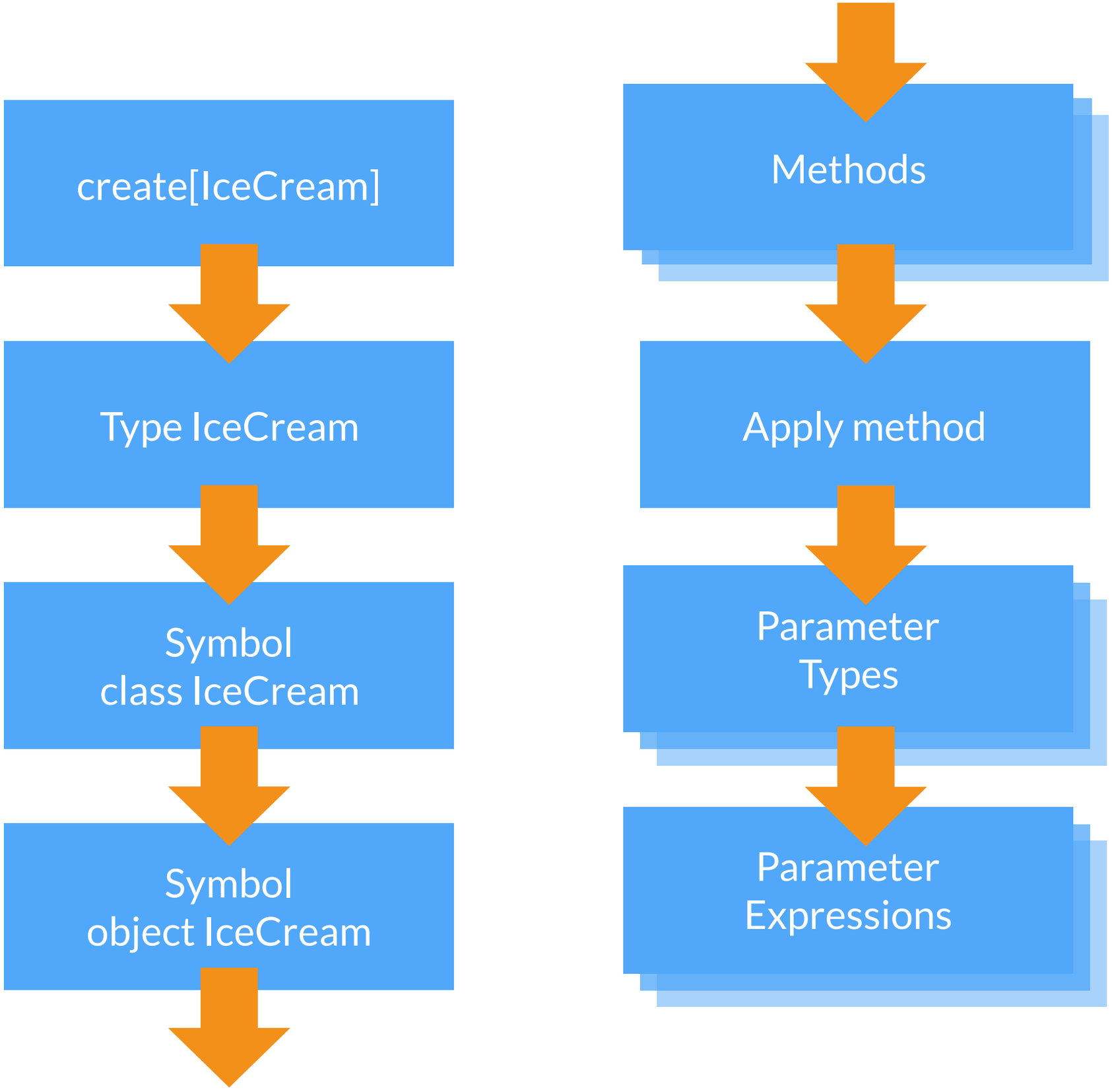
macros

```
create[IceCream]
```

```
IceCream("", 0, false)
```

```
def create[A]: A =
  macro Macros.createMacro[A]
```

```
def createMacro[A: WeakTypeTag]: Tree = {
  val applyMethod = ???

  val applyParams = ???

  q"$applyMethod(...$applyParams)"
}
```


create[IceCream]
Type IceCream
Symbol
class IceCream
Symbol
object IceCream
Methods
Apply method
Parameter
Types
Parameter
Expressions

```scala
class Macros(val c: blackbox.Context) {
  import c.universe._

  def createMacro[A: WeakTypeTag]: Tree = {
    val targetType = weakTypeOf[A]

    val applyMethod = findApplyMethod(targetType)

    val applyParams = applyMethod.paramLists.map { paramList =>
      paramList.map { param =>
        createApplyParam(param.typeSignature)
      }
    }

    q"$applyMethod(...$applyParams)"
  }

  def findApplyMethod(targetType: Type): MethodSymbol =
    targetType.companion
      .members
      .find { member =>
        member.isMethod &&
        member.isPublic &&
        member.asMethod.returnType <:< targetType &&
        member.asMethod.name.decodedName.toString == "apply"
      }
      .map(_.asMethod)
      .getOrElse(c.abort(c.enclosingPosition, “FAIL!"))

  def createApplyParam(paramType: Type): Tree =
    if(paramType <:< typeOf[String]) {
      q""" "" """
    } else if(paramType <:< typeOf[Int]) {
      q"0"
    } else if(paramType <:< typeOf[Boolean]) {
      q"false"
    } else {
      c.abort(c.enclosingPosition, "FAIL!")
    }
}
```

```scala
class Macros(val c: blackbox.Context) {
  import c.universe._

  def createMacro[A: WeakTypeTag]: Tree = {
    val targetType = weakTypeOf[A]

    val applyMethod = findApplyMethod(targetType)

    val applyParams = applyMethod.paramLists.map { paramList =>
      paramList.map { param =>
        createApplyParam(param.typeSignature)
      }
    }

    q"$applyMethod(...$applyParams)"
  }

  def findApplyMethod(targetType: Type): MethodSymbol =
    targetType.companion
      .members
      .find { member =>
        member.isMethod &&
        member.isPublic &&
        member.asMethod.returnType <:< targetType &&
        member.asMethod.name.decodedName.toString == "apply"
      }
      .map(_.asMethod)
      .getOrElse(c.abort(c.enclosingPosition, “FAIL!"))

  def createApplyParam(paramType: Type): Tree =
    if(paramType <:< typeOf[String]) {
      q""" "" """
    } else if(paramType <:< typeOf[Int]) {
      q"0"
    } else if(paramType <:< typeOf[Boolean]) {
      q"false"
    } else {
      c.abort(c.enclosingPosition, "FAIL!")
    }
}
```

~40 loc

```scala
class Macros(val c: blackbox.Context) {
  import c.universe._

  def createMacro[A: WeakTypeTag]: Tree = {
    val targetType = weakTypeOf[A]

    val applyMethod = findApplyMethod(targetType)

    val applyParams = applyMethod.paramLists.map { paramList =>
      paramList.map { param =>
        createApplyParam(param.typeSignature)
      }
    }

    q"$applyMethod(...$applyParams)"
  }

  def findApplyMethod(targetType: Type): MethodSymbol =
    targetType.companion
      .members
      .find { member =>
        member.isMethod &&
        member.isPublic &&
        member.asMethod.returnType <:< targetType &&
        member.asMethod.name.decodedName.toString == "apply"
      }
      .map(_.asMethod)
      .getOrElse(c.abort(c.enclosingPosition, “FAIL!"))

  def createApplyParam(paramType: Type): Tree =
    if(paramType <:< typeOf[String]) {
      q""" "" """
    } else if(paramType <:< typeOf[Int]) {
      q"0"
    } else if(paramType <:< typeOf[Boolean]) {
      q"false"
    } else {
      c.abort(c.enclosingPosition, "FAIL!")
    }
}
```

~40 loc

```scala
class Macros(val c: blackbox.Context) {
  import c.universe._

  def createMacro[A: WeakTypeTag]: Tree = {
    val targetType = weakTypeOf[A]

    val applyMethod = findApplyMethod(targetType)

    val applyParams = applyMethod.paramLists.map { paramList =>
      paramList.map { param =>
        createApplyParam(param.typeSignature)
      }
    }

    q"$applyMethod(...$applyParams)"
  }

  def findApplyMethod(targetType: Type): MethodSymbol =
    targetType.companion
      .members
      .find { member =>
        member.isMethod &&
        member.isPublic &&
        member.asMethod.returnType <:< targetType &&
        member.asMethod.name.decodedName.toString == "apply"
      }
      .map(_.asMethod)
      .getOrElse(c.abort(c.enclosingPosition, “FAIL!"))

  def createApplyParam(paramType: Type): Tree =
    if(paramType <:< typeOf[String]) {
      q""" "" """
    } else if(paramType <:< typeOf[Int]) {
      q"0"
    } else if(paramType <:< typeOf[Boolean]) {
      q"false"
    } else {
      c.abort(c.enclosingPosition, "FAIL!")
    }
}
```

~40 loc

```scala
class Macros(val c: blackbox.Context) {
  import c.universe._

  def createMacro[A: WeakTypeTag]: Tree = {
    val targetType = weakTypeOf[A]

    val applyMethod = findApplyMethod(targetType)

    val applyParams = applyMethod.paramLists.map { paramList =>
      paramList.map { param =>
        createApplyParam(param.typeSignature)
      }
    }

    q"$applyMethod(...$applyParams)"
  }

  def findApplyMethod(targetType: Type): MethodSymbol =
    targetType.companion
      .members
      .find { member =>
        member.isMethod &&
        member.isPublic &&
        member.asMethod.returnType <:< targetType &&
        member.asMethod.name.decodedName.toString == "apply"
      }
      .map(_.asMethod)
      .getOrElse(c.abort(c.enclosingPosition, “FAIL!"))

  def createApplyParam(paramType: Type): Tree =
    if(paramType <:< typeOf[String]) {
      q""" "" """
    } else if(paramType <:< typeOf[Int]) {
      q"0"
    } else if(paramType <:< typeOf[Boolean]) {
      q"false"
    } else {
      c.abort(c.enclosingPosition, "FAIL!")
    }
}
```

~40 loc

```
def createApplyParam(paramType: Type): Tree =
  if(paramType <:< typeOf[String]) {
    q""" "" """
  } else if(paramType <:< typeOf[Int]) {
    q"0"
  } else if(paramType <:< typeOf[Boolean]) {
    q"false"
  } else {
    c.abort(c.enclosingPosition, "FAIL!")
  }
```

```
create[IceCream]
```

```
IceCream("", 0, false)
```

analysis

scala-reflect is deprecated!

we should be using scala-meta!

only handles three parameter types

not customisable by the user

macros v2

```scala
trait Pure[A] {
  def value: A
}

implicit val stringPure: Pure[String] =
  new Pure[String] { def value = "" }

implicit val intPure: Pure[Int] =
  new Pure[Int] { def value = 0 }

implicit val booleanPure: Pure[Boolean] =
  new Pure[Boolean] { def value = false }
```

```scala
trait Pure[A] {
  def value: A
}

implicit val stringPure: Pure[String] =
  new Pure[String] { def value = "" }

implicit val intPure: Pure[Int] =
  new Pure[Int] { def value = 0 }

implicit val booleanPure: Pure[Boolean] =
  new Pure[Boolean] { def value = false }

implicitly[Pure[String]] // => stringPure
```

```
def createApplyParam(paramType: Type): Tree =
  q"""
  _root_.scala.Predef
    .implicitly[_root_.Pure[$paramType]]
    .value
"""
```

```scala
def createApplyParam(paramType: Type): Tree =
  q"""
  _root_.scala.Predef
    .implicitly[_root_.Pure[$paramType]]
    .value
  """

// implicitly[Pure[$paramType]].value
```

```
create[IceCream]
```

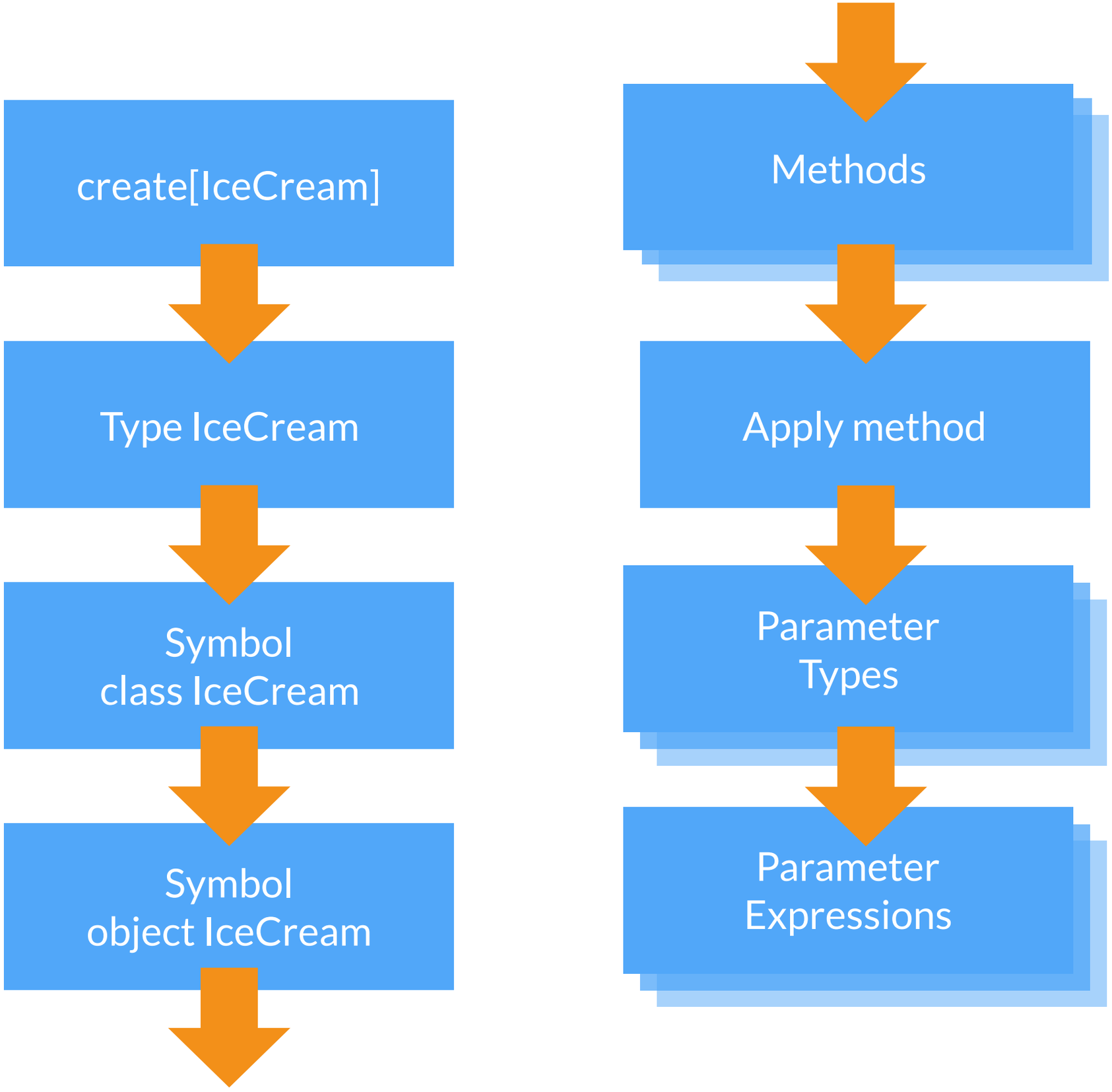
```
create[IceCream](IceCream(
  implicitly[Pure[String]].value,
  implicitly[Pure[Int]].value,
  implicitly[Pure[Boolean]].value
))
```

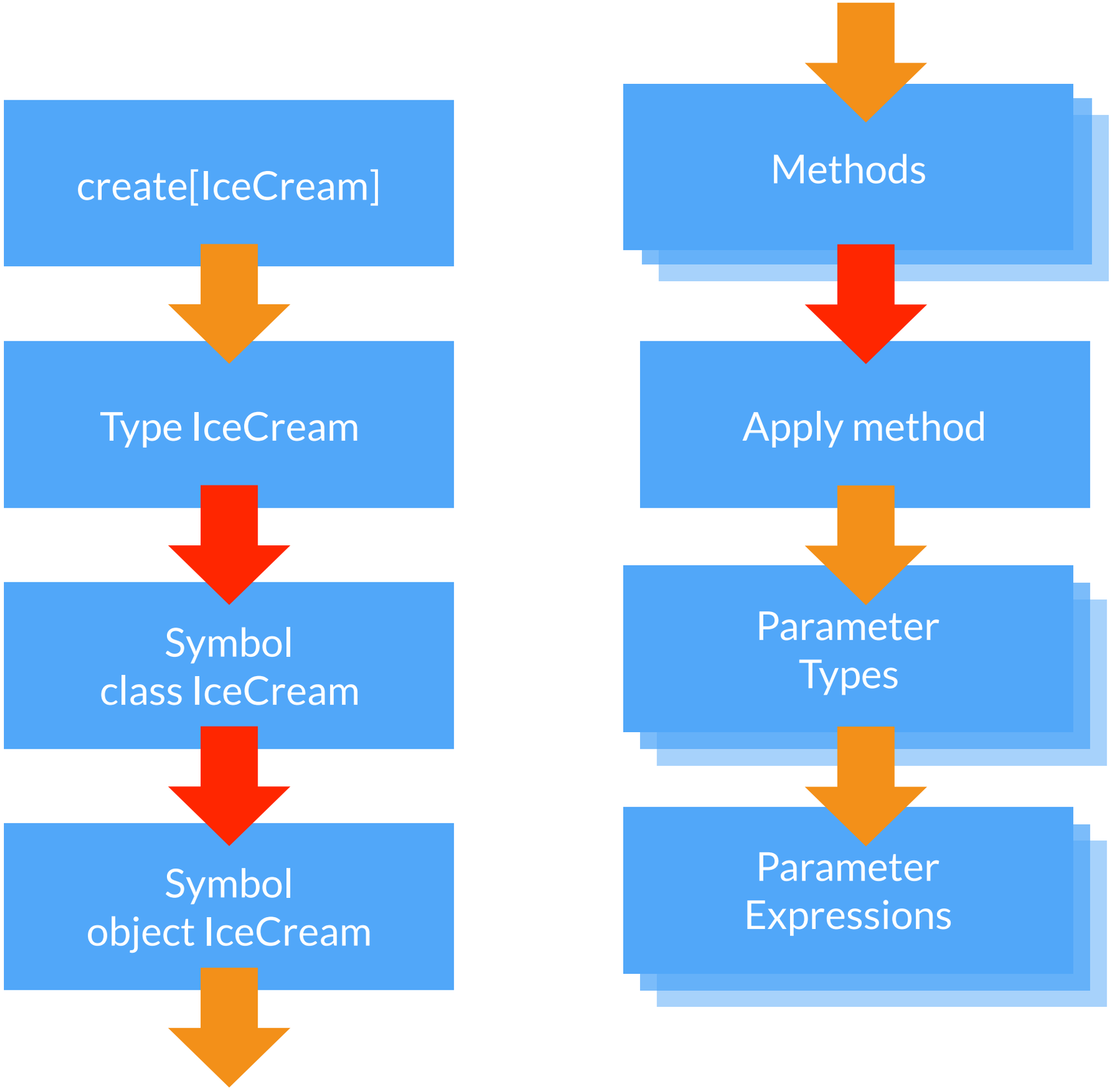
analysis

user-customisable

handles any parameter type

brittle in various ways

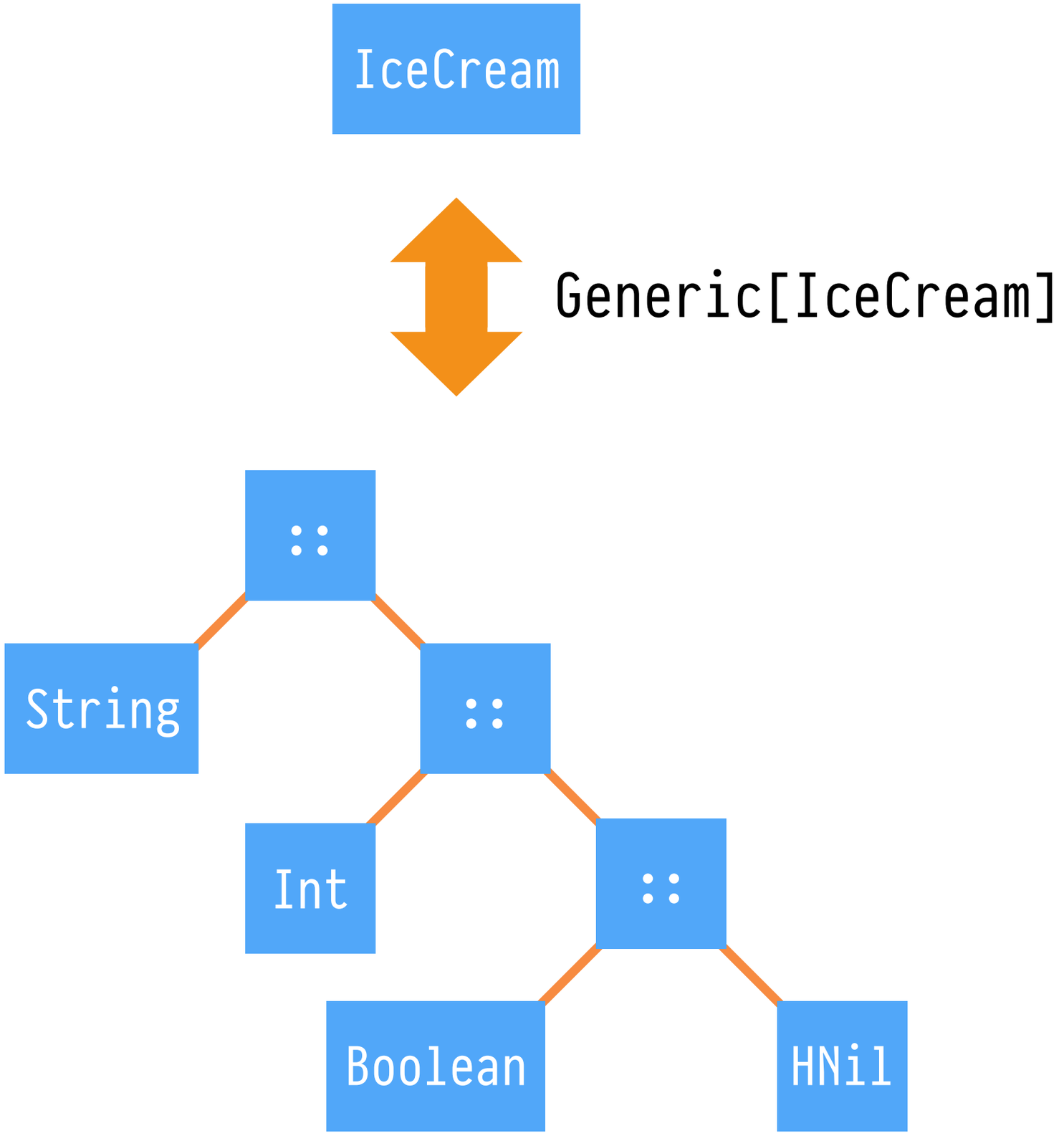

create[IceCream]
Type IceCream
Symbol
class IceCream
Symbol
object IceCream
Methods
Apply method
Parameter
Types
Parameter
Expressions

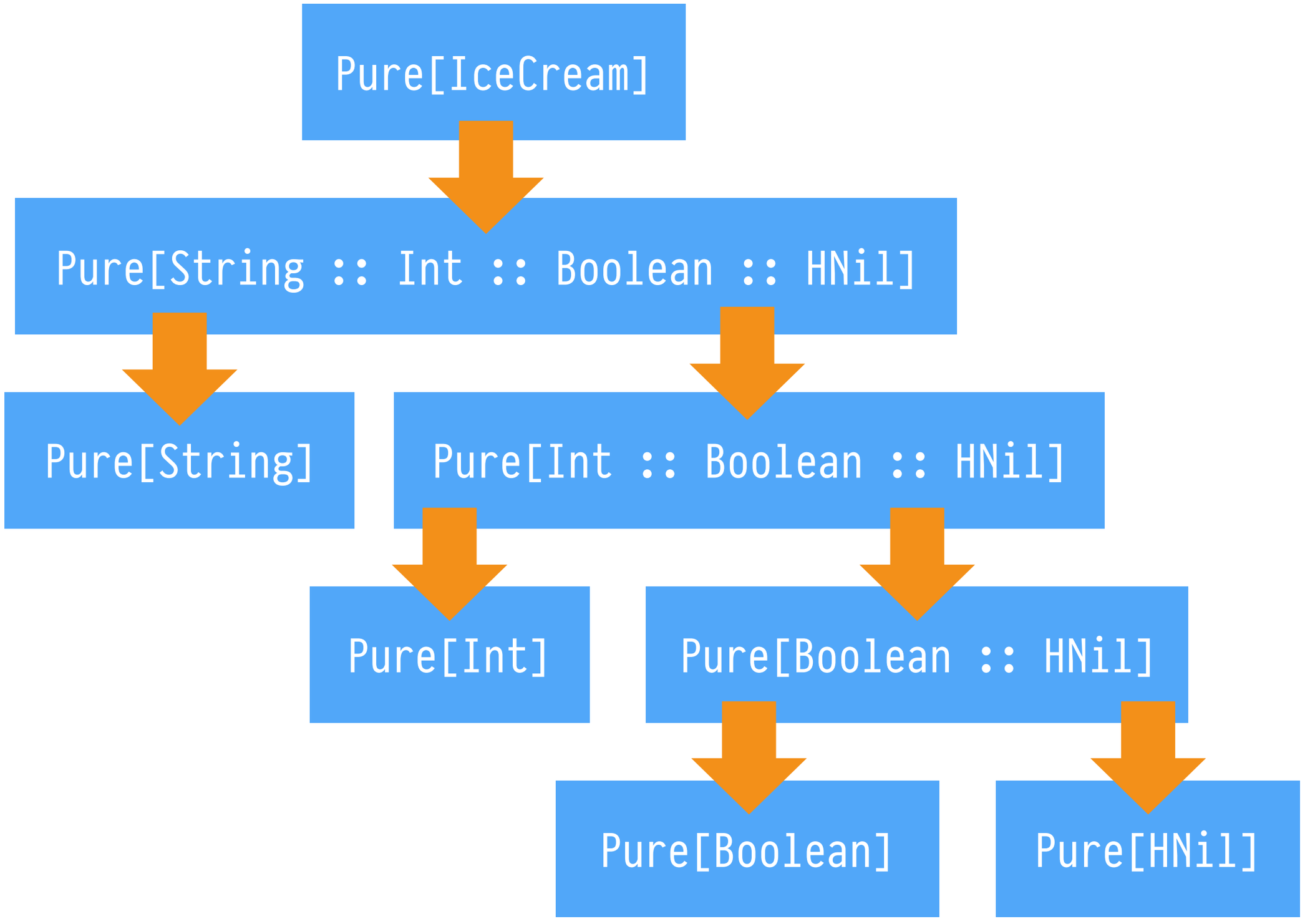

create[IceCream]
Type IceCream
Symbol
class IceCream
Symbol
object IceCream
Methods
Apply method
Parameter
Types
Parameter
Expressions

shapeless

```scala
trait Pure[A] {
  def value: A
}

implicit val stringPure: Pure[String] =
  new Pure[A] { def value = "" }

implicit val intPure: Pure[Int] =
  new Pure[A] { def value = 0 }

implicit val booleanPure: Pure[Boolean] =
  new Pure[A] { def value = false }
```

```scala
trait Pure[A] {
  def value: A
}

implicit val stringPure: Pure[String] =
  new Pure[A] { def value = "" }

implicit val intPure: Pure[Int] =
  new Pure[A] { def value = 0 }

implicit val booleanPure: Pure[Boolean] =
  new Pure[A] { def value = false }

implicit def genericPure[A]: Pure[A] =
  ???
```


IceCream
Generic[IceCream]
::
String
::
Int
::
Boolean
HNil


Pure[IceCream]
Pure[String :: Int :: Boolean :: HNil]
Pure[String]
Pure[Int :: Boolean :: HNil]
Pure[Int]
Pure[Boolean :: HNil]
Pure[Boolean]
Pure[HNil]

```scala
implicit val hnilPure: Pure[HNil] =
  instance(HNil)

implicit def hconsPure[H, T <: HList](
  implicit
  hPure: Lazy[Pure[H]],
  tPure: Pure[T]
): Pure[H :: T] =
  instance(hPure.value.value :: tPure.value)

implicit def genericPure[A, R](
  implicit
  gen: Generic.Aux[A, R],
  pure: Lazy[Pure[R]]
): Pure[A] =
  instance(gen.from(pure.value.value))
```

```scala
implicit val hnilPure: Pure[HNil] =
  instance(HNil)

implicit def hconsPure[H, T <: HList](
  implicit
  hPure: Lazy[Pure[H]],
  tPure: Pure[T]
): Pure[H :: T] =
  instance(hPure.value.value :: tPure.value)

implicit def genericPure[A, R](
  implicit
  gen: Generic.Aux[A, R],
  pure: Lazy[Pure[R]]
): Pure[A] =
  instance(gen.from(pure.value.value))
```

```scala
implicit val hnilPure: Pure[HNil] =
  instance(HNil)

implicit def hconsPure[H, T <: HList](
  implicit
  hPure: Lazy[Pure[H]],
  tPure: Pure[T]
): Pure[H :: T] =
  instance(hPure.value.value :: tPure.value)

implicit def genericPure[A, R](
  implicit
  gen: Generic.Aux[A, R],
  pure: Lazy[Pure[R]]
): Pure[A] =
  instance(gen.from(pure.value.value))
```

```scala
implicit val hnilPure: Pure[HNil] =
  instance(HNil)

implicit def hconsPure[H, T <: HList](
  implicit
  hPure: Lazy[Pure[H]],
  tPure: Pure[T]
): Pure[H :: T] =
  instance(hPure.value.value :: tPure.value)

implicit def genericPure[A, R](
  implicit
  gen: Generic.Aux[A, R],
  pure: Lazy[Pure[R]]
): Pure[A] =
  instance(gen.from(pure.value.value))
```

```scala
implicit val hnilPure: Pure[HNil] =
  instance(HNil)

implicit def hconsPure[H, T <: HList](
  implicit
  hPure: Lazy[Pure[H]],
  tPure: Pure[T]
): Pure[H :: T] =
  instance(hPure.value.value :: tPure.value)

implicit def genericPure[A, R](
  implicit
  gen: Generic.Aux[A, R],
  pure: Lazy[Pure[R]]
): Pure[A] =
  instance(gen.from(pure.value.value))
```

~16 loc

analysis

shorter, more maintainable

both approaches use
similar techniques

however:
error messages,
compile times!

error messages

error messages

```
<console>:12: error: could not find implicit value
  for parameter e: Pure[IceCream]
       implicitly[Pure[IceCream]]
                  ^
```

compile times

compile times

```
case class BigData(
  a: Int, b: Int, c: Int, d: Int, e: Int,
f: Int, g: Int, h: Int, i: Int, j: Int,
k: Int, l: Int, m: Int, n: Int, o: Int,
p: Int, q: Int, r: Int, s: Int, t: Int,
u: Int, v: Int, w: Int, x: Int, y: Int,
z: Int)
```

compile times

```
case class BigData(
  a: Int, b: Int, c: Int, d: Int, e: Int,
f: Int, g: Int, h: Int, i: Int, j: Int,
k: Int, l: Int, m: Int, n: Int, o: Int,
p: Int, q: Int, r: Int, s: Int, t: Int,
u: Int, v: Int, w: Int, x: Int, y: Int,
z: java.util.Date)
```

compile times

```
case class Outer(a: Middle, b: Middle, c: Middle)
case class Middle(a: Inner, b: Inner, c: Inner)
case class Inner(a: String, b: String, c: String)
```

compile times

```
case class Outer(a: Middle, b: Middle, c: Middle)
case class Middle(a: Inner, b: Inner, c: Inner)
case class Inner(a: String, b: String, c: String)

implicit val innerPure: Pure[Inner] = cachedImplicit
implicit val middlePure: Pure[Middle] = cachedImplicit
implicit val outerPure: Pure[Outer] = cachedImplicit
```

what wins?

in this case, IMO shapeless

shapeless' `Generic` isolates
the meta-programming

"all" we do is write implicits!

macwire

```
import macwire._

val database = Database()
val routing  = Routing()

val service  = wire[Service]
```

```
import macwire._

val database = Database()
val routing  = Routing()

val service  = Service(database, routing)
```

```
import macwire._

val database = Database()
val routing  = Routing()

val service  = Service(database, routing)
```

~350 loc

data validation

https://github.com/davegurnell/checklist

```
case class IceCream(
  name: String,
  cherries: Int,
  cone: Boolean)
```

```scala
case class IceCream(
  name: String,
  cherries: Int,
  cone: Boolean)
```

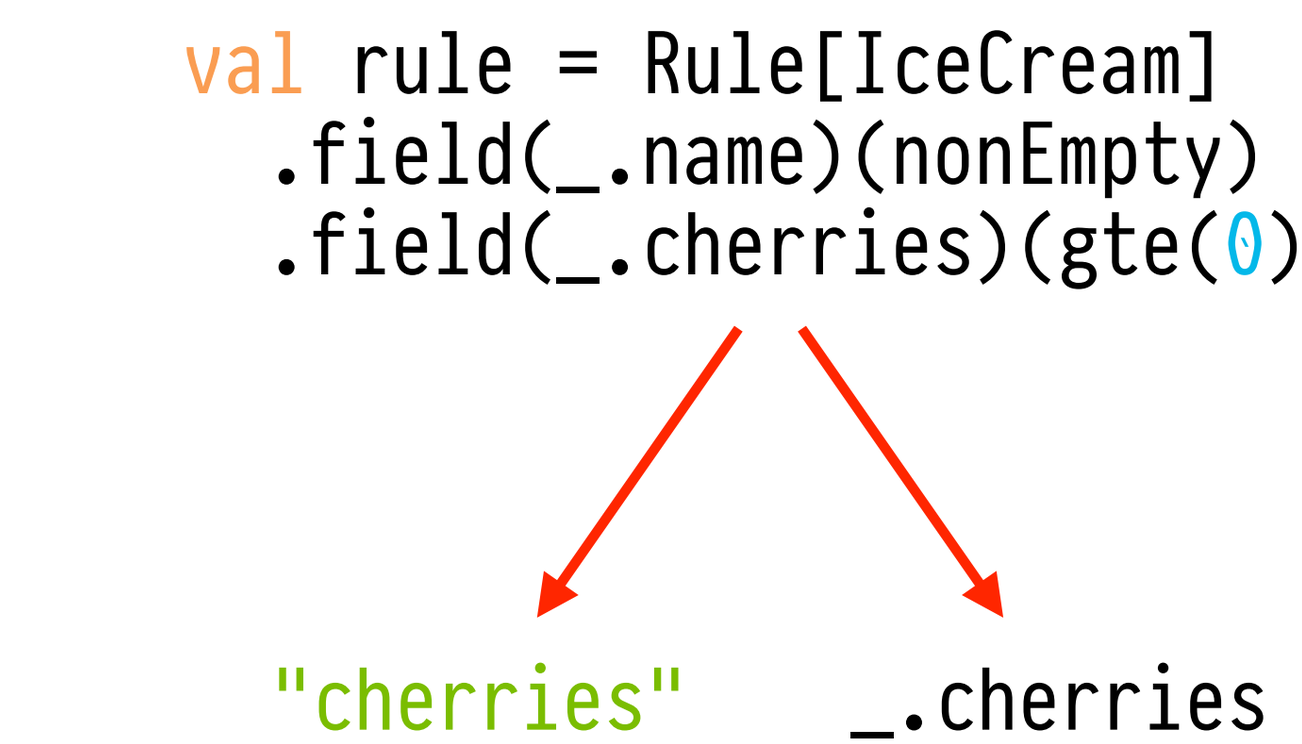
```scala
val rule = Rule[IceCream]
  .field(_.name)(nonEmpty)
  .field(_.cherries)(gte(0))
```

```scala
case class IceCream(
  name: String,
  cherries: Int,
  cone: Boolean)

val rule = Rule[IceCream]
  .field(_.name)(nonEmpty)
  .field(_.cherries)(gte(0))

rule(IceCream("", -1, false))
// List(
//   Error("Must be non-empty", List("name")),
//   Error("Must be >= 0",      List("cherries"))
// ))
```

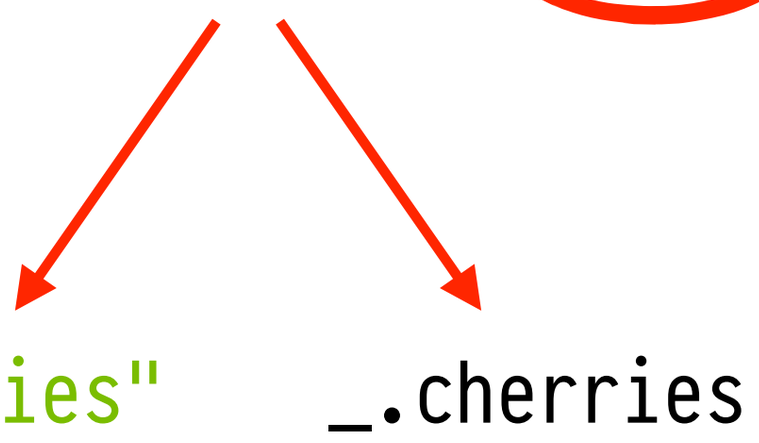
```scala
case class IceCream(
  name: String,
  cherries: Int,
  cone: Boolean)

val rule = Rule[IceCream]
  .field(_.name)(nonEmpty)
  .field(_.cherries)(gte(0))

rule(IceCream("", -1, false))
// List(
//   Error("Must be non-empty", List("name")),
//   Error("Must be >= 0",      List("cherries"))
// ))
```

macros

```
val rule = Rule[IceCream]
  .field(_.name)(nonEmpty)
  .field(_.cherries)(gte(0))
```



```
val rule = Rule[IceCream]
  .field(_.name)(nonEmpty)
  .field(_.cherries)(gte(0))
```



```
val rule = gte(0)
rule.apply(-1)
// List(Error("Must be >= 0", Nil))
```

```
val rule = gte(0)
  .prefixed("cherries")

rule.apply(-1)
// List(Error("Must be >= 0", List("cherries")))
```

```scala
val rule = gte(0)
  .prefixed("cherries")
  .contramap[IceCream](_.cherries)
```

```scala
rule.apply(IceCream("Sundae", -1, false))
// List(Error("Must be >= 0", List("cherries")))
```

```scala
val rule = Rule[IceCream]
  .field(_.name)(nonEmpty)
  .field(_.cherries)(gte(0))
```

```scala
val rule = Rule[IceCream]
  .and(nonEmpty
    .prefixed("name")
    .contramap[IceCream](_.name))
  .and(gte(0)
    .prefixed("cherries")
    .contramap[IceCream](_.cherries))
```

```
trait Rule[A] {
  def apply(value: A): List[Error]

  def field(func: A => B)(rule: Rule[B]): Rule[A] =
    macro Macros.fieldMacro(func)(rule)
}
```

```scala
class Macros(val c: blackbox.Context) {
  import c.universe._

  def fieldMacro(func: Tree)(rule: Tree): Tree = {
    val name = func match {
      case q"($param) => $obj.$name" =>
        q"${name.toString}"
      case other =>
        c.abort(c.enclosingPosition, "FAIL!")
    }

    q"""
    ${c.prefix}.and(
      $rule.prefixed($name).contramap($func))
    """
  }
}
```

~16 loc

```
Rule[IceCream]
  .field(_.cherries)(gte(0))
```

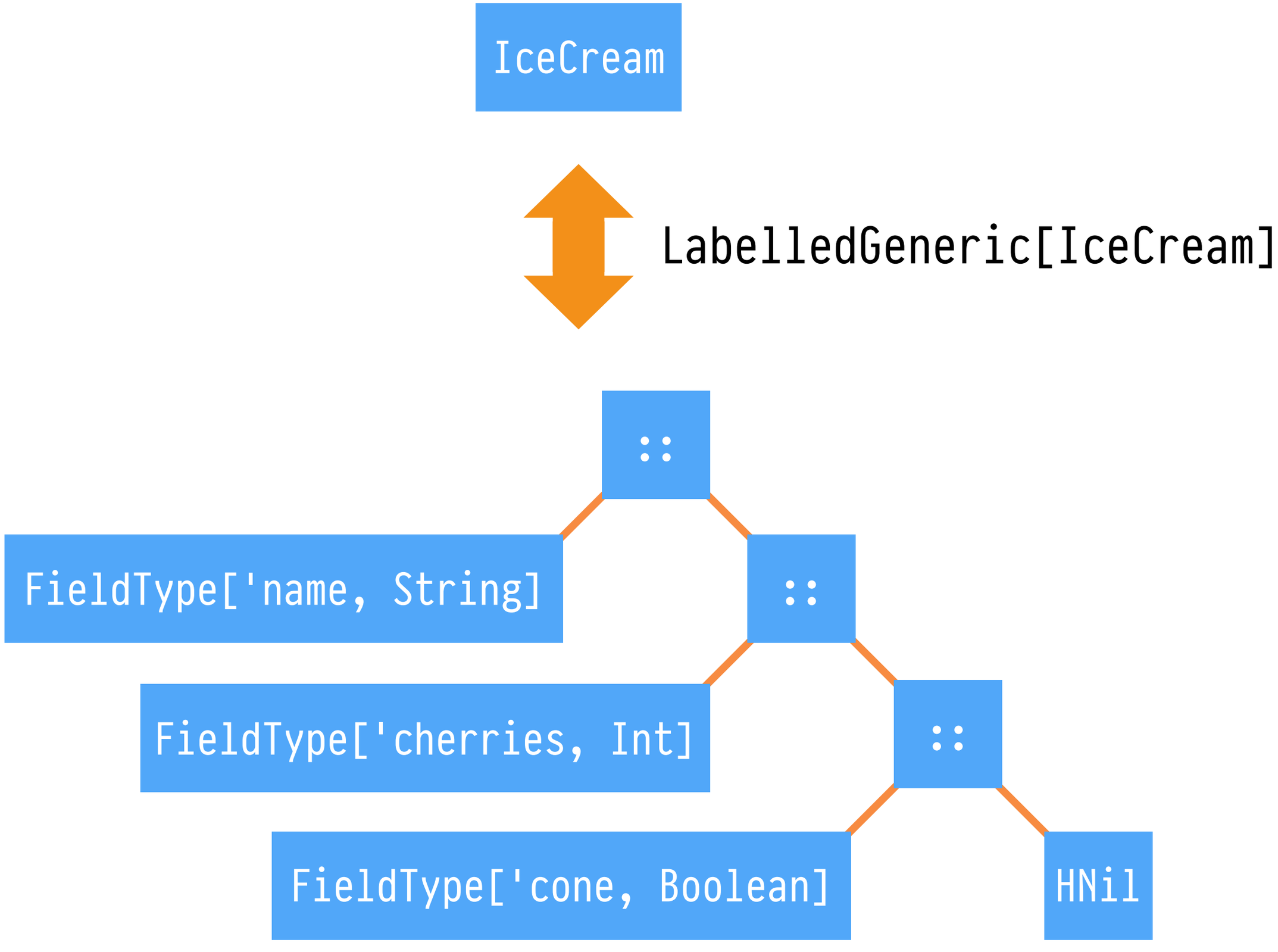
```
Rule[IceCream].and(
  get(0)
    .prefixed("cherries")
    .contramap[IceCream](_.cherries))
```

analysis

short and simple

syntactic solution
to a syntactic problem

shapeless


IceCream
LabelledGeneric[IceCream]
::
FieldType['name, String]
::
FieldType['cherries, Int]
::
FieldType['cone, Boolean]
HNil

```
FieldType['name, String]
```

field name
(literal type)

field type

```
val field1: Symbol = 'name
val field2: 'name  = 'name
```

```
val number1: Int = 42
val number2: 42  = 42
```

```
val rule = Rule[IceCream]
  .field('name)(nonEmpty)
  .field('cherries)(gte(0))
```

```scala
trait Rule[A] {
  def apply(value: A): List[Error]

  def field[B]
      (field: Witness)
      (rule: Rule[B])
      (implicit wrap: HasField[A, field.T, B]): Rule[A] =
    this.and(wrap(rule))
}

trait HasField[A, K, B] {
  def name: String
  def zoom(value: A): B

  def apply(rule: Rule[B]): Rule[A] =
    rule.prefixed(name).contramap(zoom)
}

implicit def hlistHasField[L <: HList, K, F](
  implicit
  ev: K <:< Symbol,
  witness: Witness.Aux[K],
  selector: Selector.Aux[L, K, F]
): HasField[L, K, F] =
  new HasField[L, K, F] {
    val name = witness.value.name
    def zoom(value: L): F = selector(value)
  }

implicit def genericHasField[A, L, K, F](
  implicit
  ev: K <:< Symbol,
  gen: LabelledGeneric.Aux[A, L],
  hf: HasField[L, K, F]
): HasField[A, K, F] =
  new HasField[A, K, F] {
    val name = hf.name
    def zoom(value: A): F = hf.zoom(gen.to(value))
  }
```

~40 loc

what wins?

in this case, IMO macros

shorter,
easier to maintain

syntactic problem,
syntactic solution

monocle

```
val lens = GenLens[IceCream](_.cherries)
```

```
val lens = Lens[IceCream, Int](_.cherries)
  (c => i => i.copy(cherries = c))
```

```
@Lenses
case class IceCream(
name: String,
cherries: Int,
cone: Boolean)
```

```scala
case class IceCream(
  name: String,
  cherries: Int,
  cone: Boolean)

object IceCream {
  val name = GenLens[IceCream](_.name)
  val cherries = GenLens[IceCream](_.cherries)
  val cone = GenLens[IceCream](_.cone)
}
```

summary

macros are good
for syntactic stuff

shapeless is good
for structural stuff

caveats

macros

brittleness/complexity

running the type checker
(especially in macro annotations)

shapeless

error messages

compile times

cachedImplicit

-Yinductive-implicits (in TLS 2.11.9+)

90% regular scala

10% meta-programming

meta-programming
is a convenience
not a solution

further reading

macros

Dave Gurnell - Macros for the Rest of Us
https://www.youtube.com/watch?v=ZVYdiAudr-I

Tomer Gabel - Leveraging Scala Macros for Better Validation
https://www.youtube.com/watch?v=Li19Cif7uS8

Chris Birchall - Meta-Program and/or Shapeless all the Things!
https://skillsmatter.com/skillscasts/9294

shapeless

Dave Gurnell - The Type Astronaut's Guide to Shapeless
http://underscore.io/books/shapeless-guide

Dave Gurnell - Establishing Orbit with Shapeless
https://skillsmatter.com/skillscasts/9136

Sam Halliday - Shapeless for Mortals
https://skillsmatter.com/skillscasts/6875

functional design

Noel Welsh - Six Core Principles for Learning Scala
https://www.youtube.com/watch?v=J8wUy1XxL5o

Dave Gurnell - Functional Data Validation (Part 1)
https://skillsmatter.com/skillscasts/5837

Dave Gurnell - Functional Data Validation (Part 2)
https://www.youtube.com/watch?v=0DPGpyt6joE

thank you

https://github.com/davegurnell/macros-vs-shapeless

bonus macro!

unindent

https://github.com/davegurnell/unindent

```
val lorem =
  """
  Lorem ipsum
  dolor sit amet
  consectetur
  """

println(lorem)
// "  Lorem ipsum\n  dolor sit\n  consectetur"
```

```
val lorem =
  """
  Lorem ipsum
  dolor sit amet
  consectetur
  """

println(lorem)
// "  Lorem ipsum\n  dolor sit\n  consectetur"
```

```scala
val lorem =
  """
    |Lorem ipsum
    |dolor sit amet
    |consectetur
  """.trim.stripMargin

println(lorem)
// "Lorem ipsum\ndolor sit\nconsectetur"
```

```scala
import unindent._

val lorem =
  i"""
  Lorem ipsum
  dolor sit amet
  consectetur
  """

println(lorem)
// "Lorem ipsum\ndolor sit\nconsectetur"
```

thank you

https://github.com/davegurnell/macros-vs-shapeless