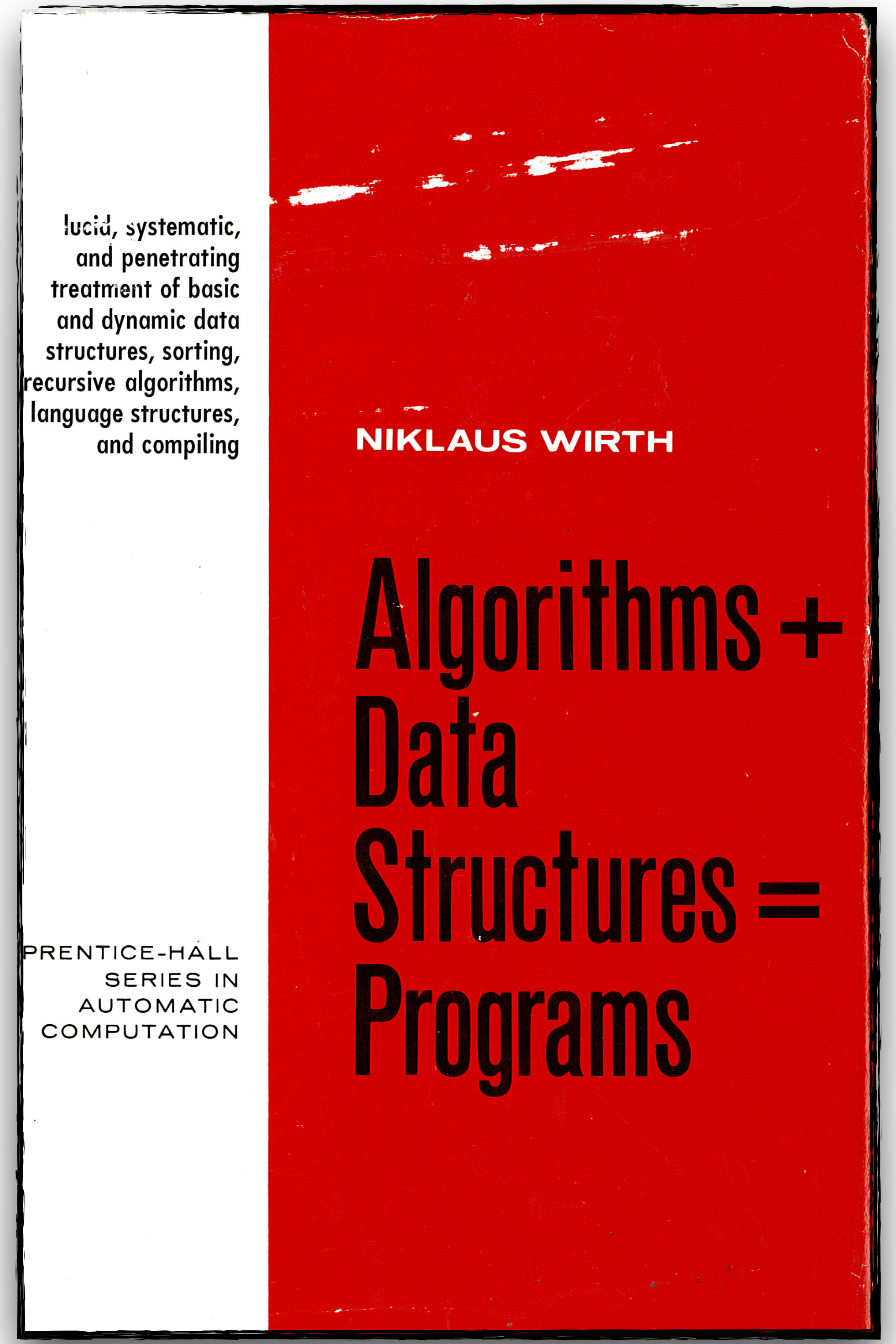
Extensibility: traits

class *or* trait *or* object	`extends`	class *or* trait	`with`	trait	`with`	trait …

data & behavior

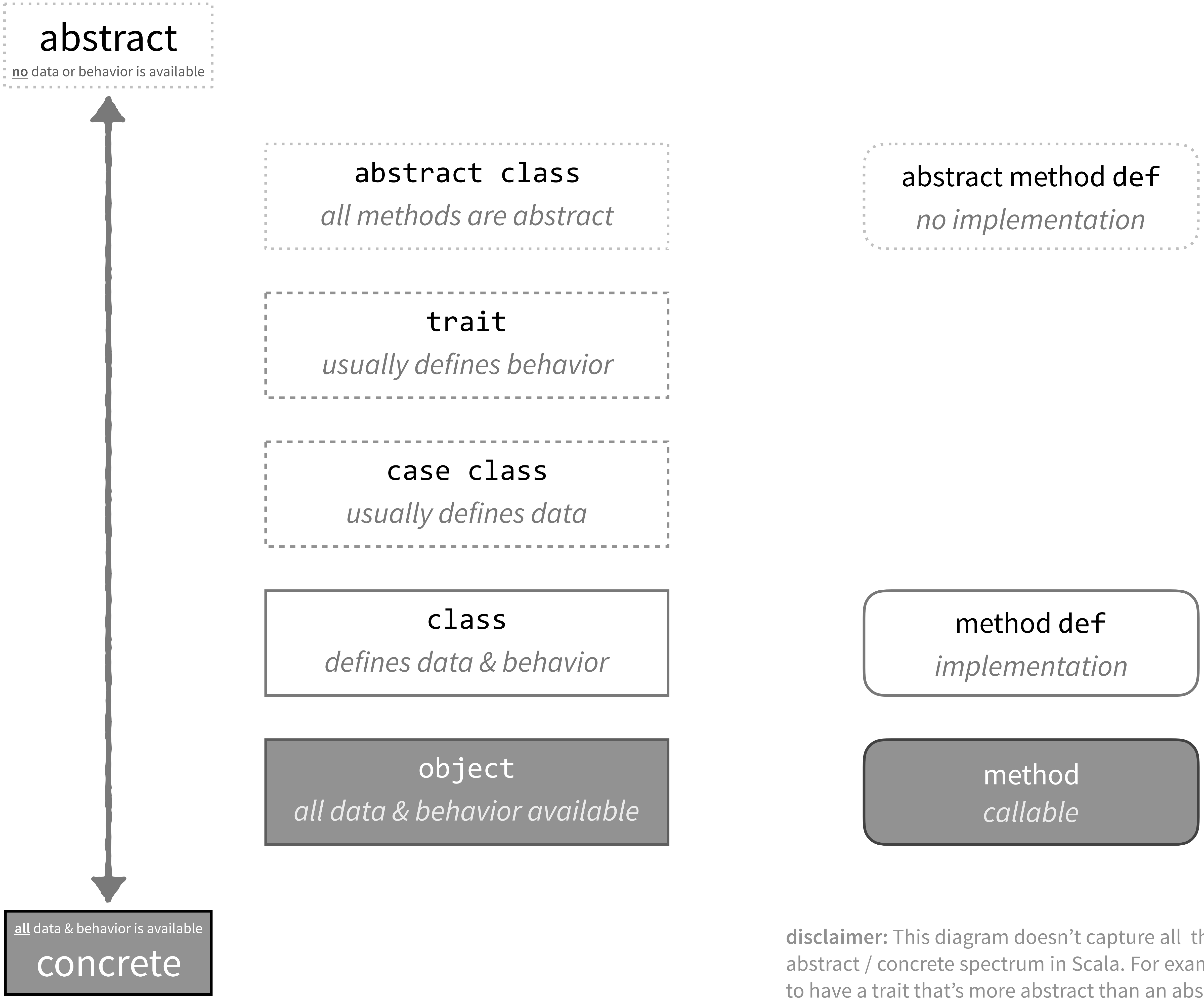


lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling

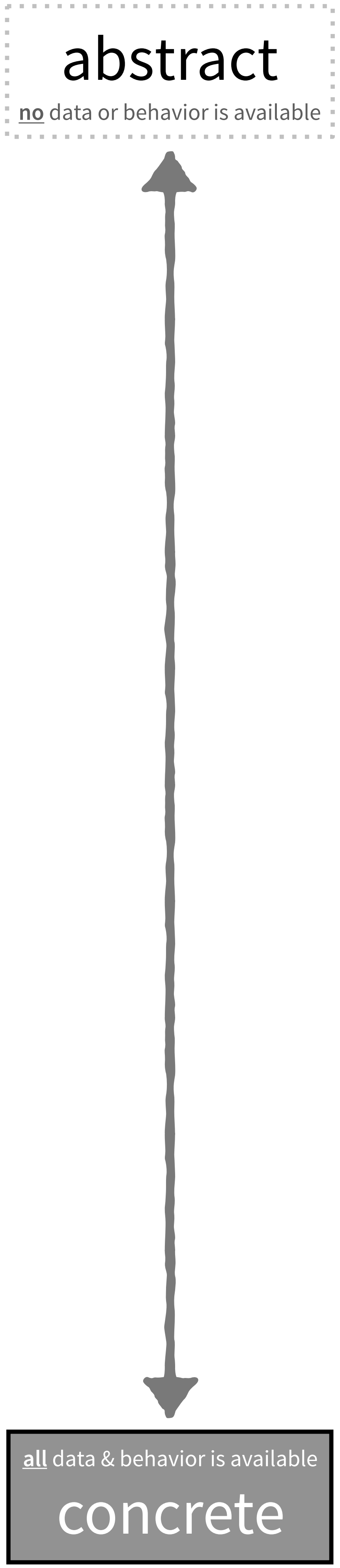
NIKLAUS WIRTH

Algorithms + Data Structures = Programs

PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION



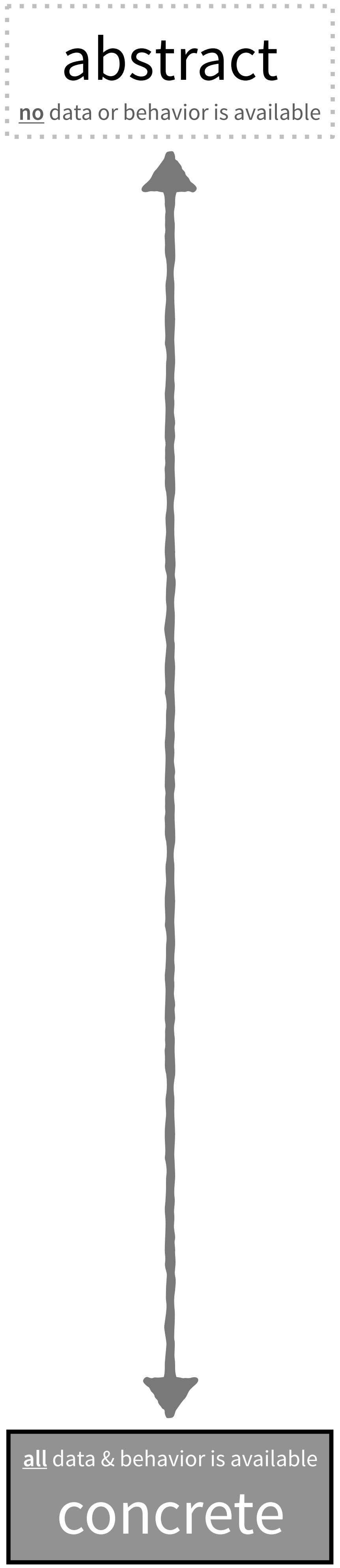
disclaimer: This diagram doesn't capture all the nuances of the abstract / concrete spectrum in Scala. For example, it's possible to have a trait that's more abstract than an abstract class.


abstract
no data or behavior is available
all data & behavior is available
concrete

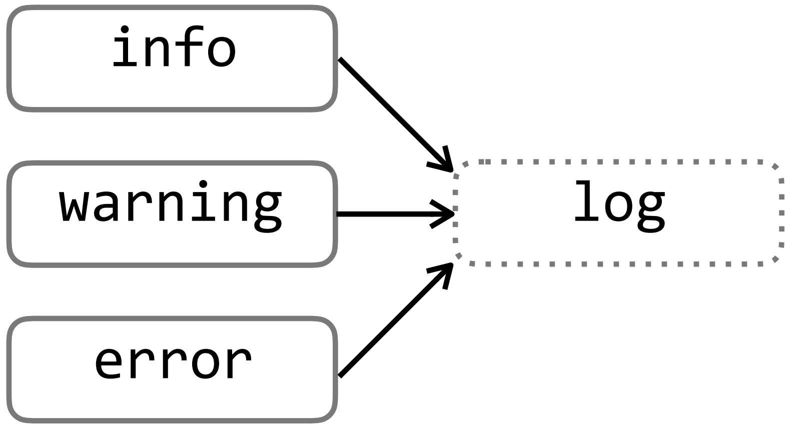
```
trait Logger {
  def log(message: String): Unit

  def infoTag    = "[info]"
  def warningTag = "[warning]"
  def errorTag   = "[error]"

  def info(message: String) = log(s"$infoTag $message")
  def warning(message: String) = log(s"$warningTag $message")
  def error(message: String) = log(s"$errorTag $message")
}
```

```
class ConsoleLogger extends Logger {
  override def log(message: String) {
    println(message)
  }
}
```

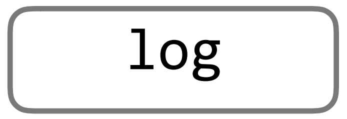
```
val logger = new ConsoleLogger()
```


abstract
no data or behavior is available
all data & behavior is available
concrete

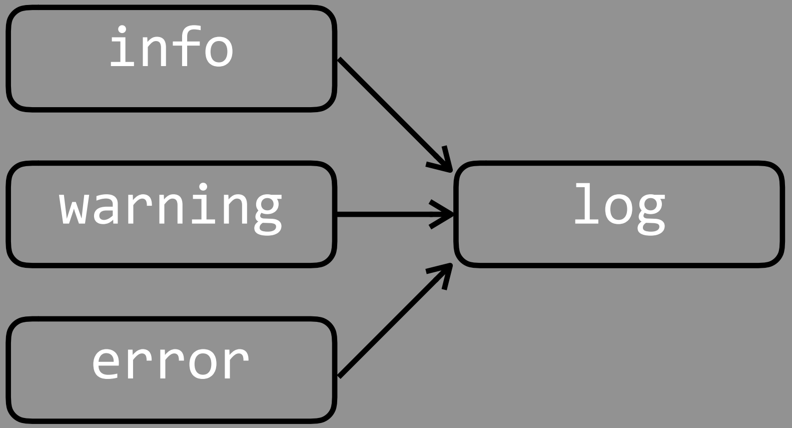
```
trait Logger {
  def log(message: String): Unit

  def infoTag    = "[info]"
  def warningTag = "[warning]"
  def errorTag   = "[error]"

  def info(message: String) = log(s"$infoTag $message")
  def warning(message: String) = log(s"$warningTag $message")
  def error(message: String) = log(s"$errorTag $message")
}
```

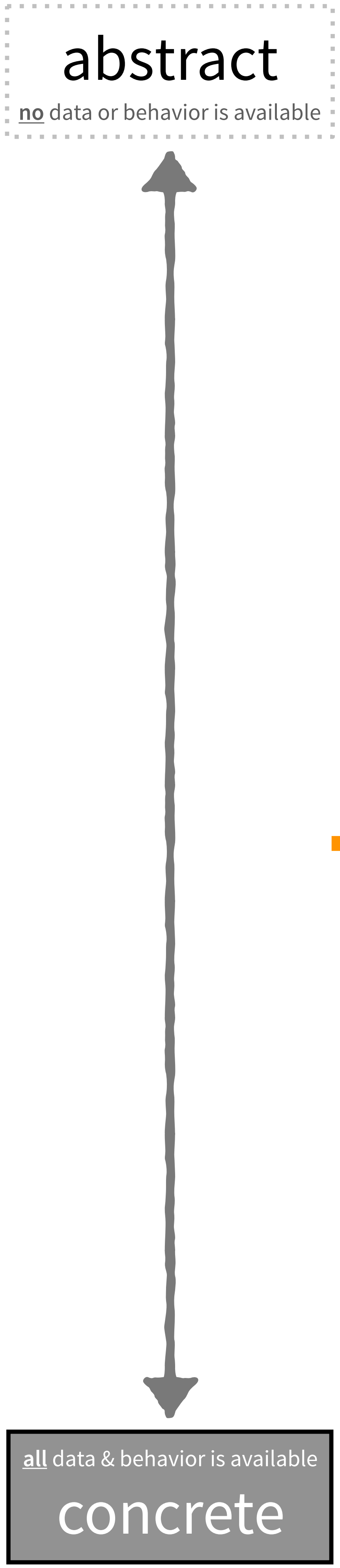


```
class ConsoleLogger extends Logger {
  override def log(message: String) {
    println(message)
  }
}
```

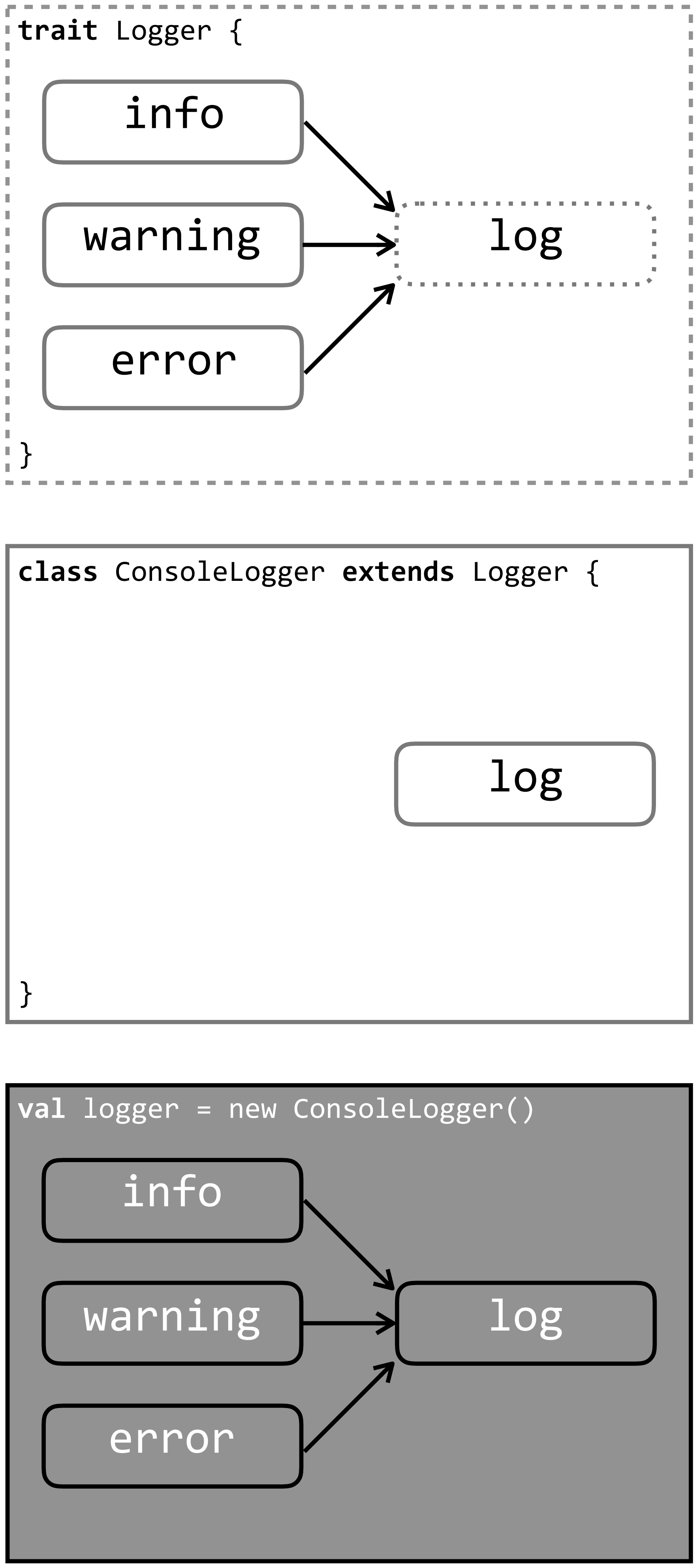


```
val logger = new ConsoleLogger()
```




abstract
no data or behavior is available
all data & behavior is available
concrete

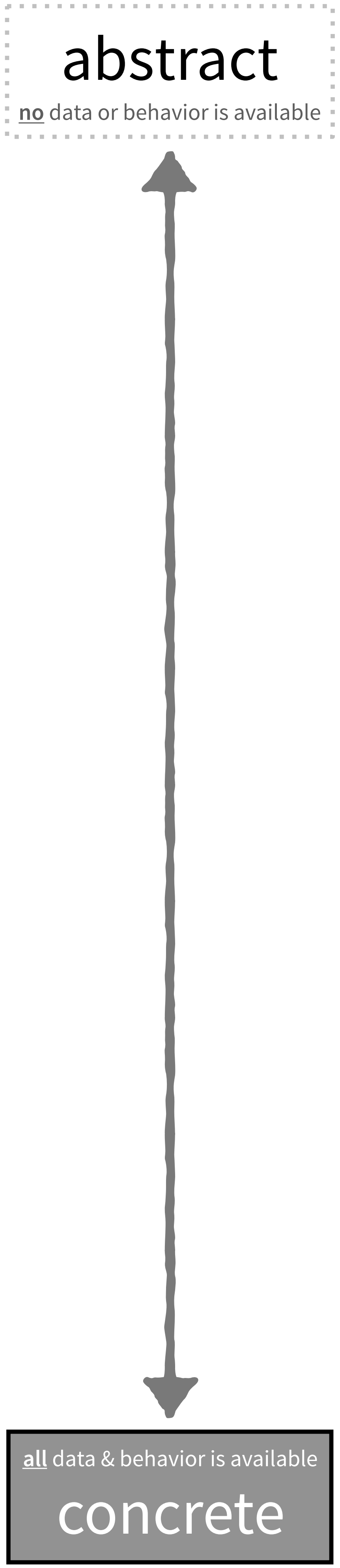
Pattern:
Thin Interface



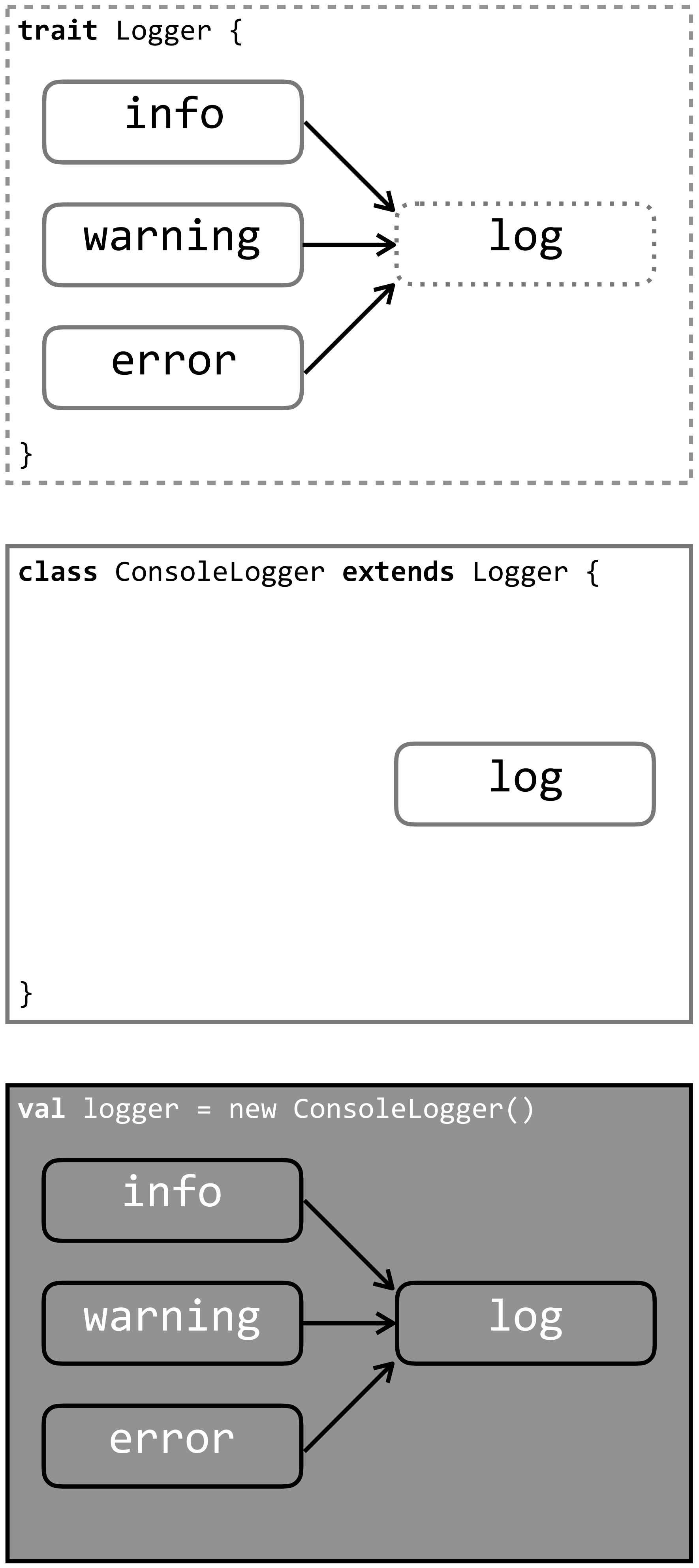
```
trait Logger {
}
```

```
class ConsoleLogger extends Logger {
}
```

```
val logger = new ConsoleLogger()
```


abstract
no data or behavior is available
all data & behavior is available
concrete

let's add some more...



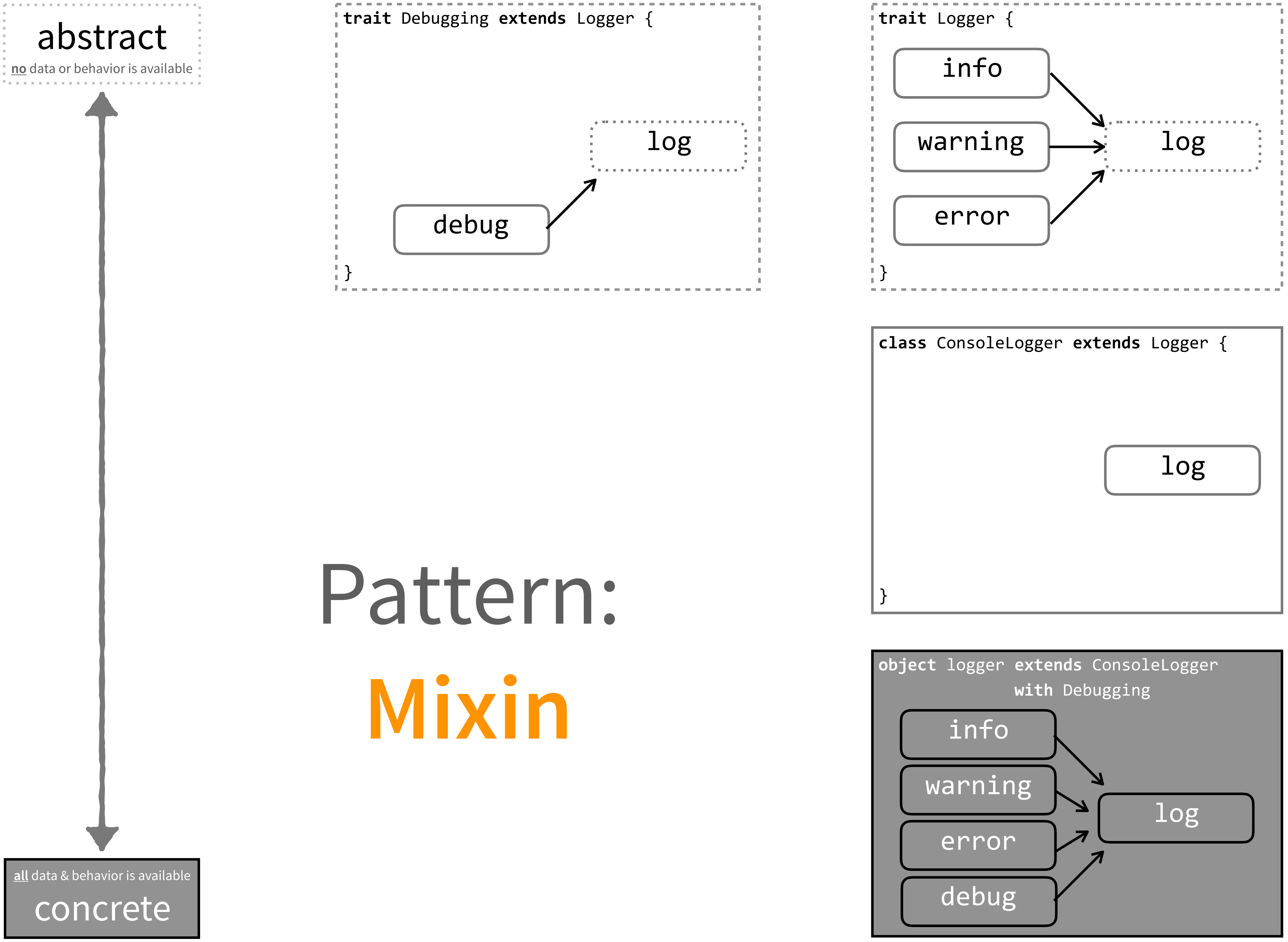
```
trait Logger {
}
```

```
class ConsoleLogger extends Logger {
}
```

```
val logger = new ConsoleLogger()
```

Pattern:

Mixin

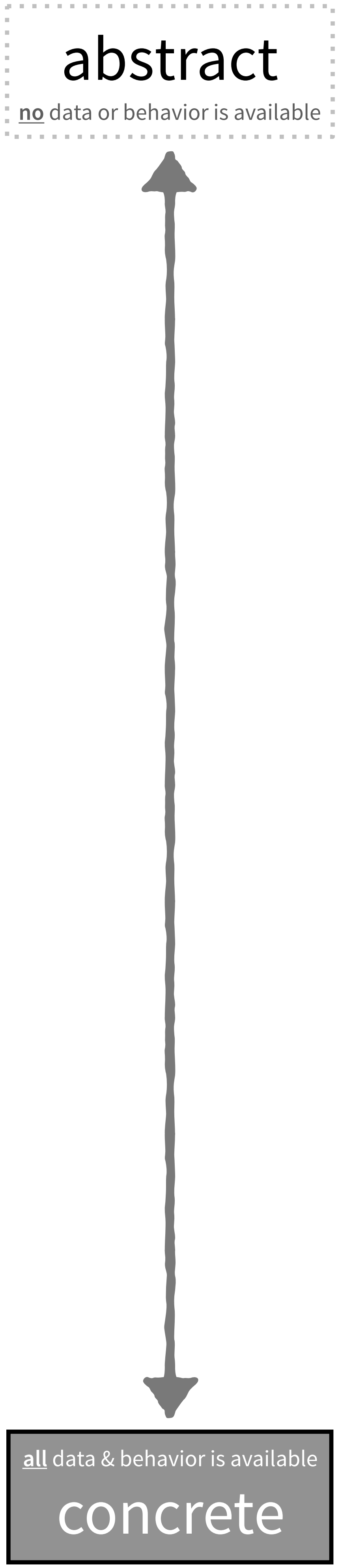


```
trait Debugging extends Logger {
}
```

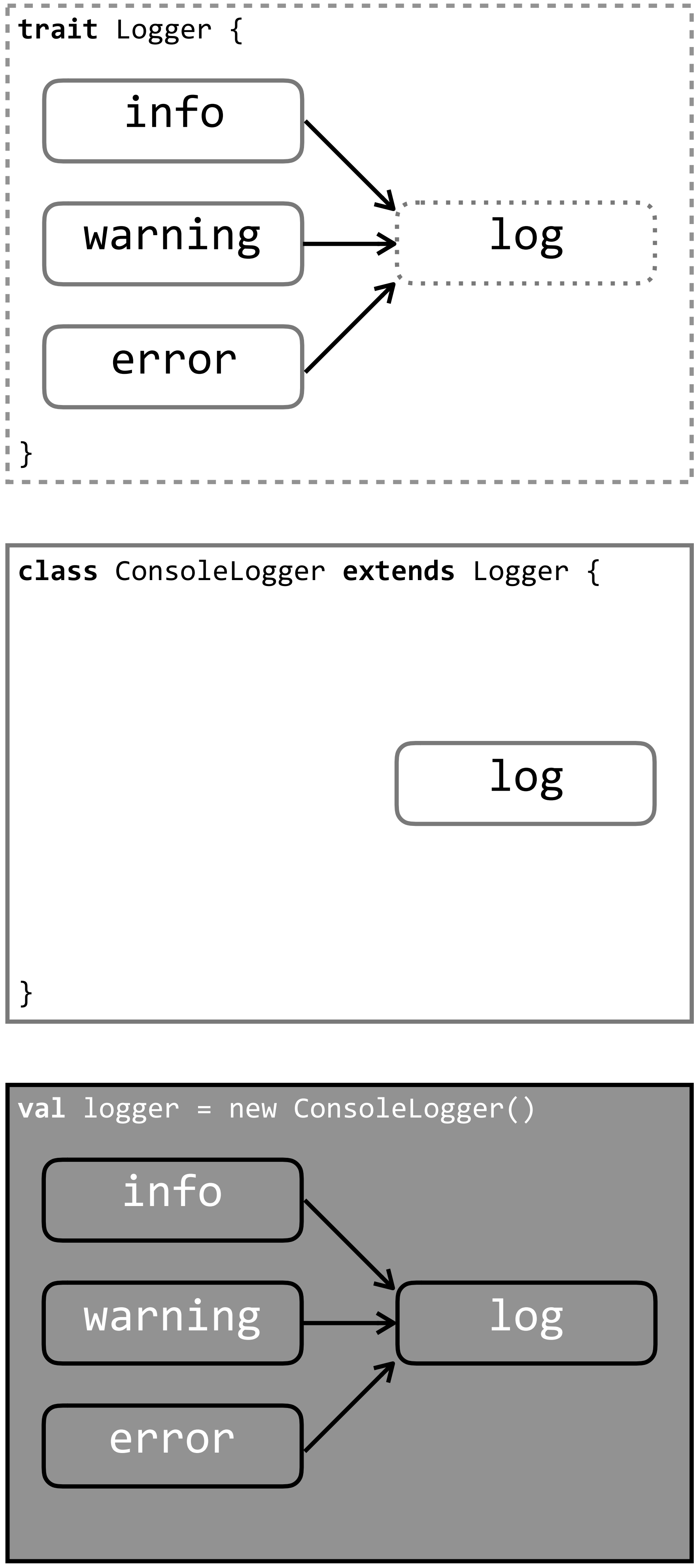
```
trait Logger {
}
```

```
class ConsoleLogger extends Logger {
}
```

```
object logger extends ConsoleLogger
              with Debugging
```

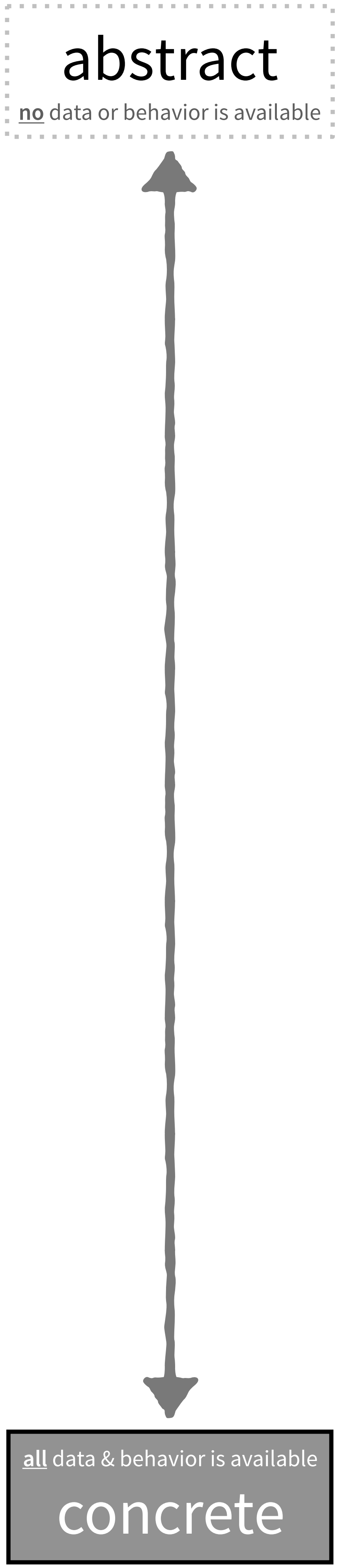

abstract
no data or behavior is available
all data & behavior is available
concrete

let's add some more...


```
trait Logger {
}
```

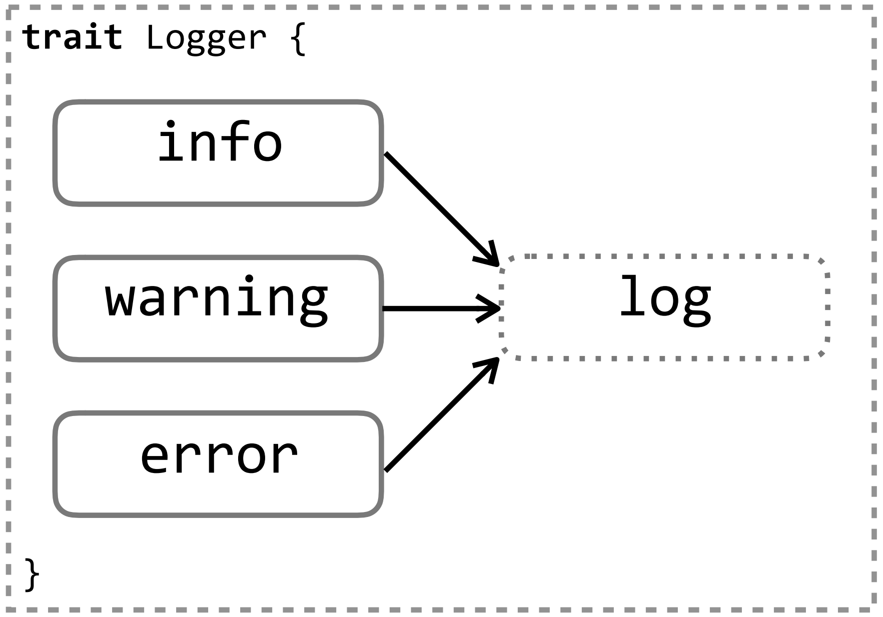
```
class ConsoleLogger extends Logger {
}
```

```
val logger = new ConsoleLogger()
```


abstract
no data or behavior is available
all data & behavior is available
concrete

```
trait Timestamping extends Logger {
  def timestamp = new java.util.Date()

  override def log(message: String) =
    s"[$timestamp] $message"
}
```

log

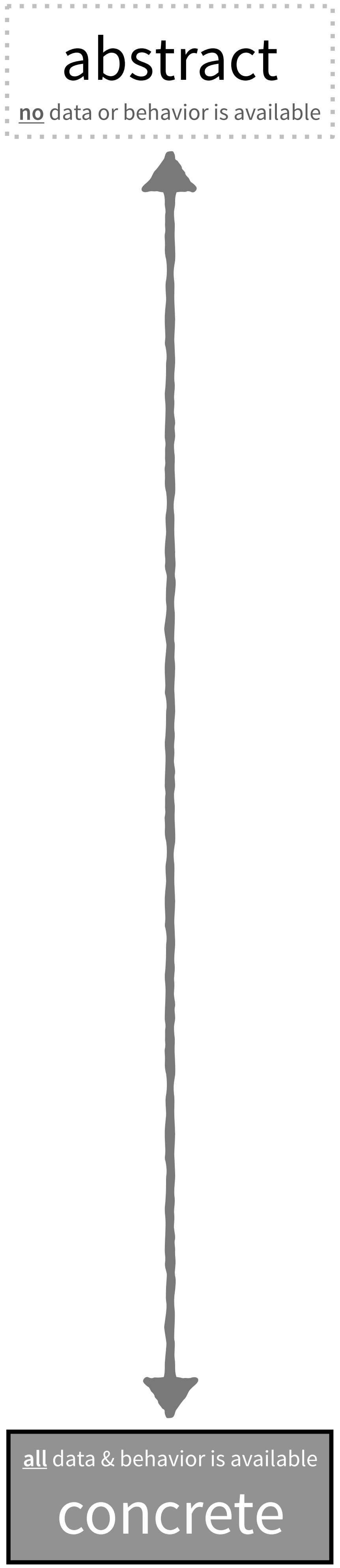


```
trait Logger {
}
```

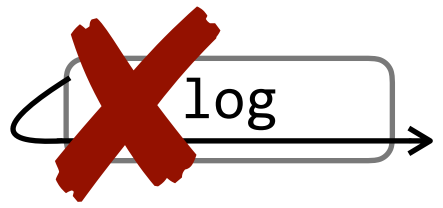
```
class ConsoleLogger extends Logger {
}
```

log

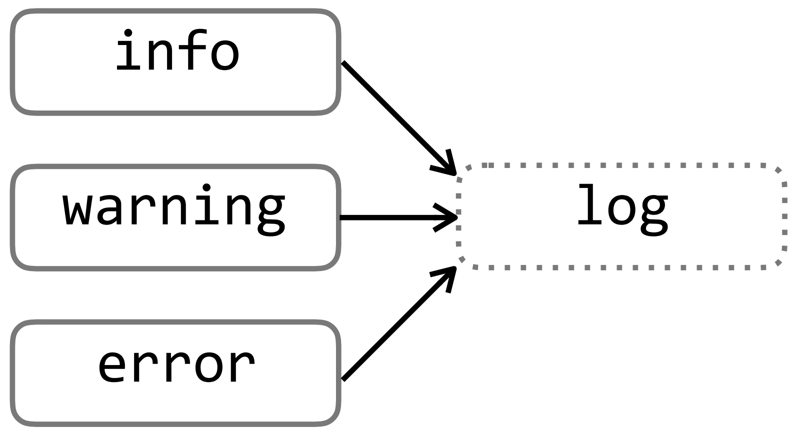
```
object logger extends ConsoleLogger
                with Timestamping
```


abstract
no data or behavior is available
all data & behavior is available
concrete

```scala
trait Timestamping extends Logger {
  def timestamp = new java.util.Date()

  override def log(message: String) =
    super.log(s"[$timestamp] $message")
}
```

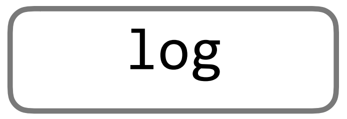


```scala
trait Logger {
```



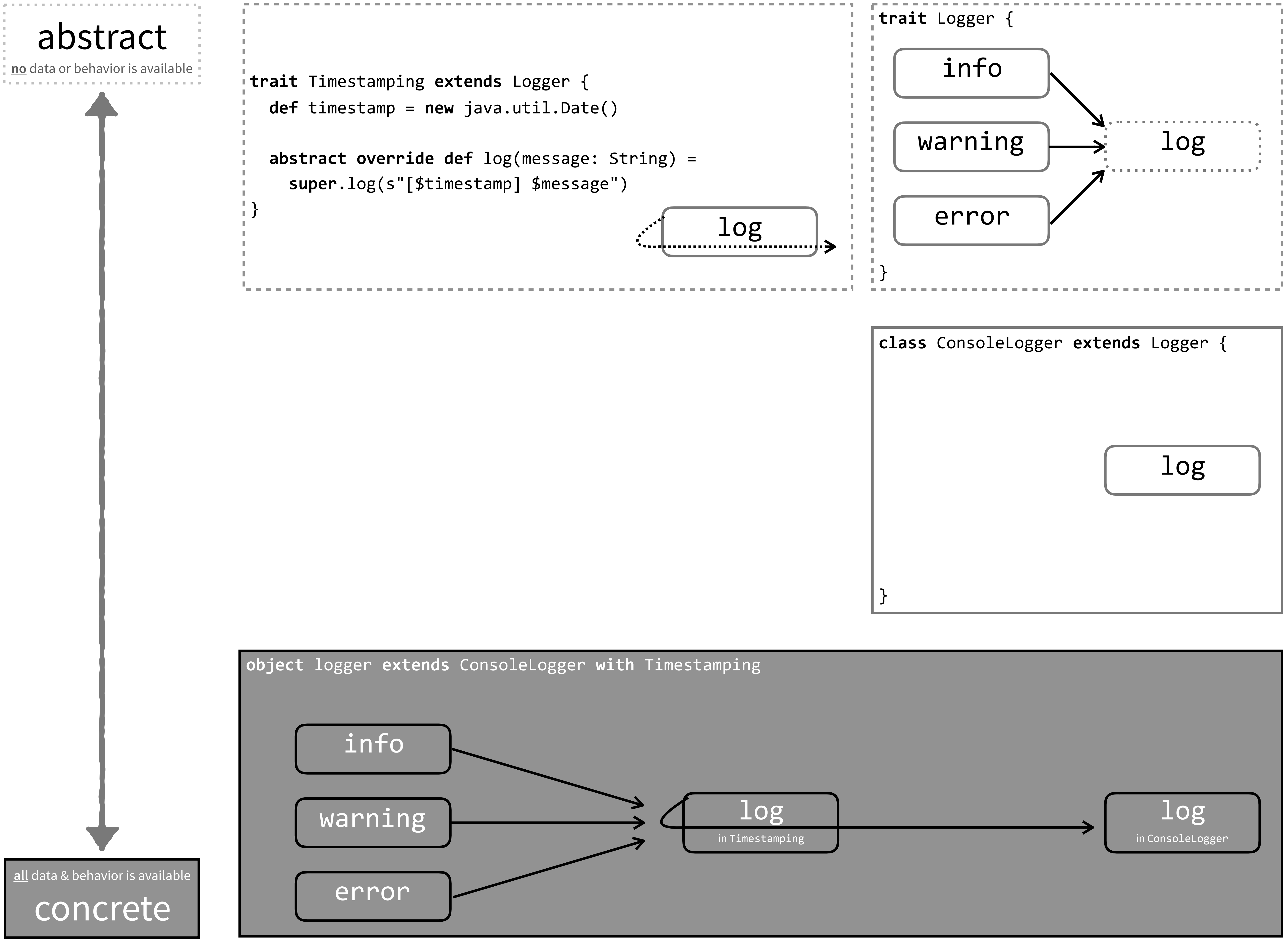
```scala
}
```

```scala
class ConsoleLogger extends Logger {
```



```scala
}
```

```scala
object logger extends ConsoleLogger
                with Timestamping
```

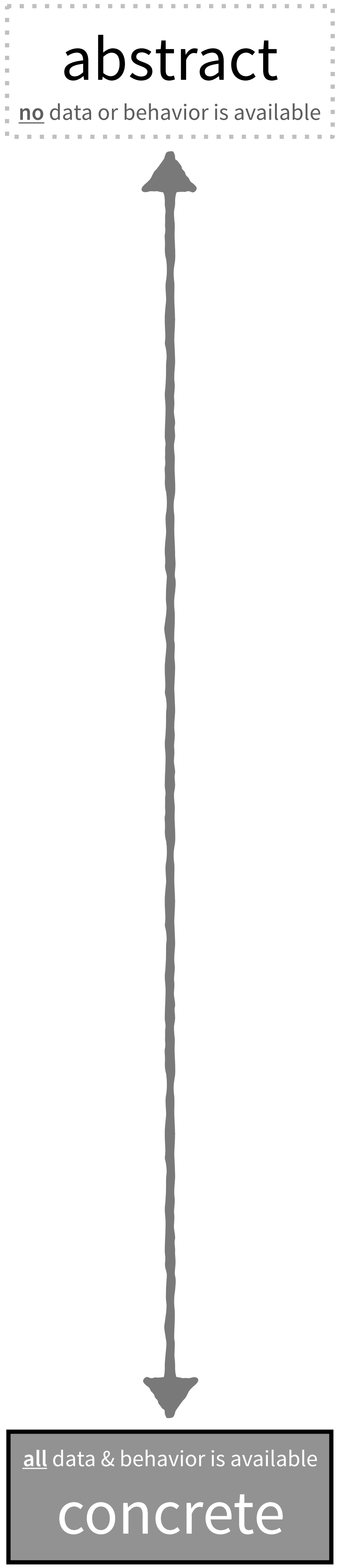


```
trait Timestamping extends Logger {
  def timestamp = new java.util.Date()

  abstract override def log(message: String) =
    super.log(s"[$timestamp] $message")
}
```

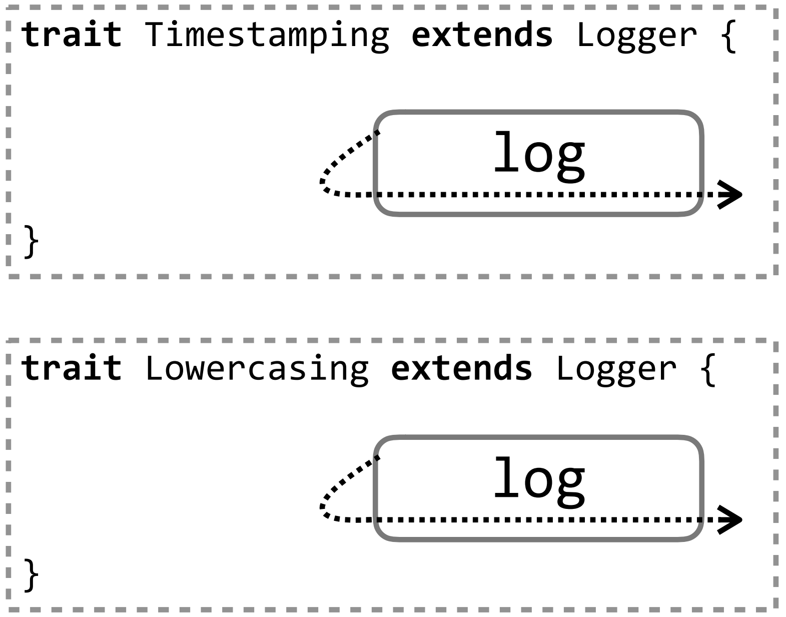
```
trait Logger {
}
```

```
class ConsoleLogger extends Logger {
}
```

```
object logger extends ConsoleLogger with Timestamping
```

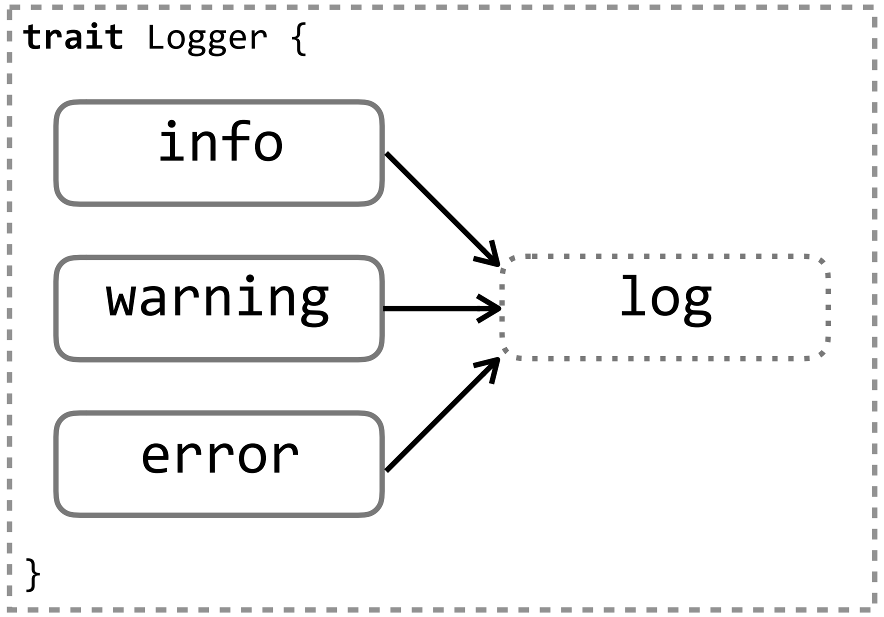

abstract
no data or behavior is available
all data & behavior is available
concrete

Pattern: Stacked Traits

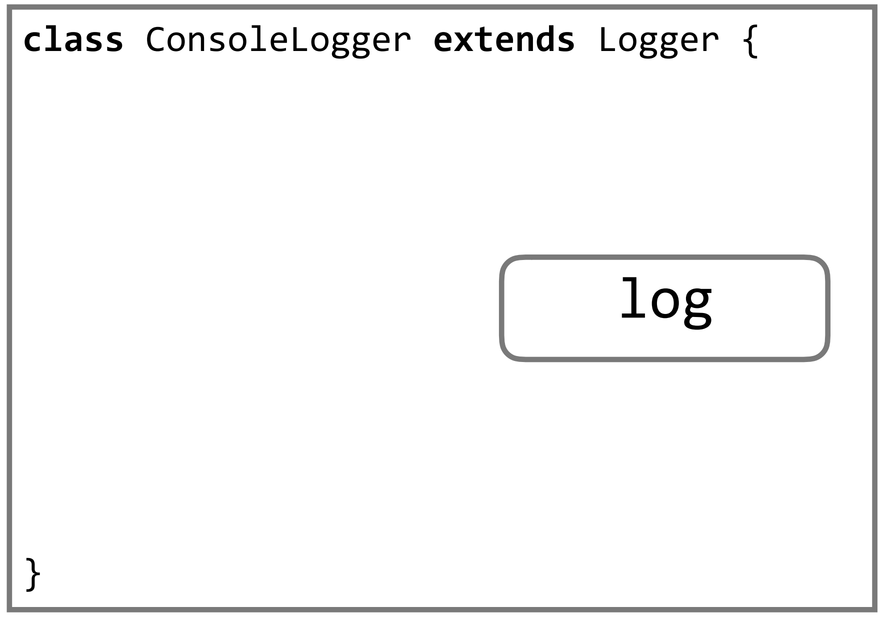


```
trait Timestamping extends Logger {
}
```

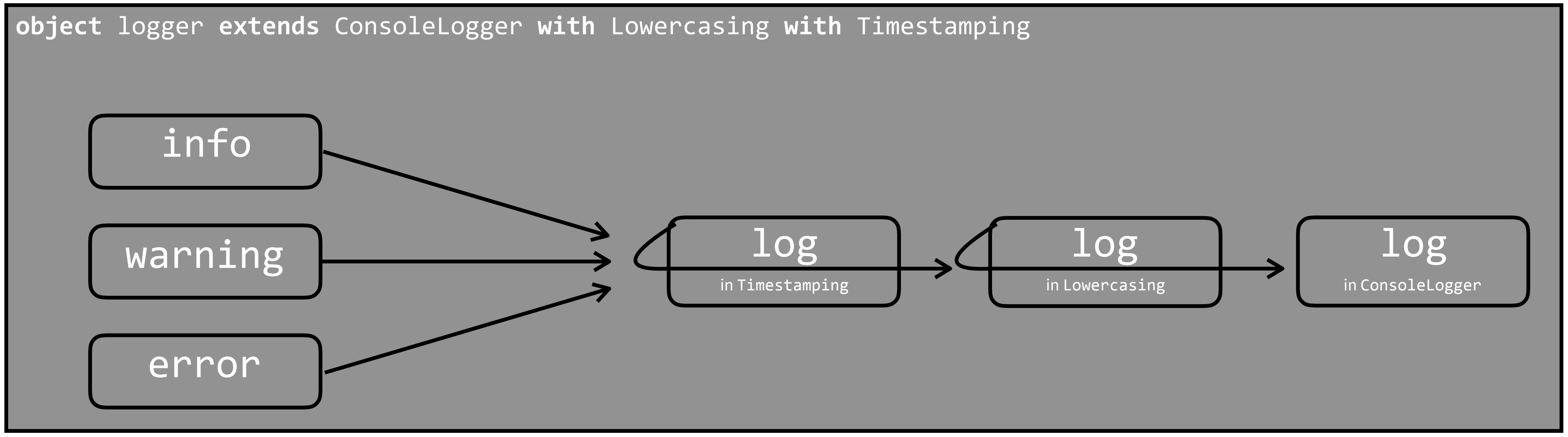
```
trait Lowercasing extends Logger {
}
```



```
trait Logger {
}
```

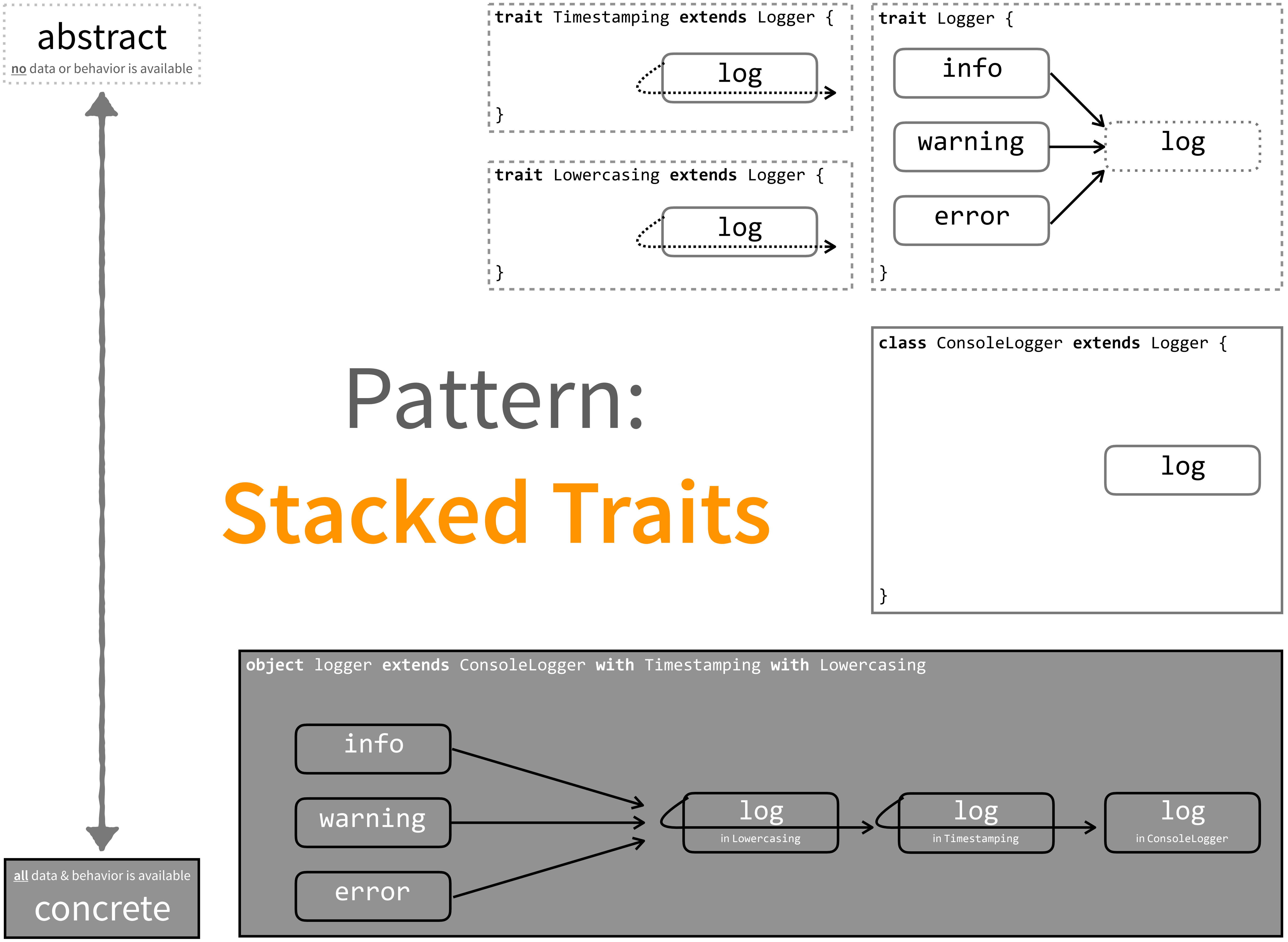


```
class ConsoleLogger extends Logger {
}
```



```
object logger extends ConsoleLogger with Lowercasing with Timestamping
```

Pattern: Stacked Traits



```
trait Timestamping extends Logger {
}

trait Lowercasing extends Logger {
}

trait Logger {
}

class ConsoleLogger extends Logger {
}

object logger extends ConsoleLogger with Timestamping with Lowercasing
```

Is this a DSL?

www.youtube.com/watch?v=rGEeLtqtNvU

- Everyone speaks with an English accent.
- Verity multiplies every number by 10.
- Lambert divides every number by 3.
- Never say “mattress” to Lambert; say “dog kennel” instead.

So, if Verity wants to say, *“This mattress costs £ 90”* to Lambert,
it will come out as *“This dog kennel costs £ 900”* (in an English accent).