

# Ad hoc Polymorphism

# Use case

```
def show[A](value: A): String =  
  value match {  
    case x: String      => x  
    case x: Double      => truncate(2, x)  
    case x: List[String] => x.mkString(",")  
    case x: List[Double] => x.map(truncate(2, _)).mkString(",")  
    case _               => "N/A"  
  }
```

# Solution 1: Overloaded functions

```
def show(value: String): String =  
    value  
  
def show(value: Double): String =  
    truncate(2, value)  
  
def defaultShow[A](value: A): String =  
    "N/A"
```

```
show("Hello")  
// res1: String = "Hello"  
show(123.123456)  
// res2: String = "123.12"  
defaultShow(true)  
// res3: String = "N/A"
```



# Solution 2: Union types (Dotty)

```
def show(value: String | Double): String =  
  value match {  
    case x: String => x  
    case x: Double => truncate(2, x)  
  }  
  
def defaultShow[A](value: A): String =  
  "N/A"
```

But similar type erasure issue

# Solution 2: Union types (Dotty)

```
case class Strings(value: List[String])
case class Doubles(value: List[Double])

def show(value: String | Double | Strings | Doubles): String =
  value match {
    case x: String => x
    case x: Double => truncate(2, x)
    case x: Strings => x.value.mkString(",")
    case x: Doubles => x.value.map(truncate(2, _)).mkString(",")
  }

def defaultShow[A](value: A): String =
  "N/A"
```

# Solution 3: Enumeration

```
sealed trait ShowValue
case class ShowString(value: String) extends ShowValue
case class ShowDouble(value: Double) extends ShowValue
case class ShowStrings(value: List[String]) extends ShowValue
case class ShowDoubles(value: List[Double]) extends ShowValue
case class ShowDefault[A](value: A) extends ShowValue

def show(value: ShowValue): String =
  value match {
    case ShowString(x)    => x
    case ShowDouble(x)    => truncate(2, x)
    case ShowStrings(x)   => x.mkString(",")
    case ShowDoubles(x)   => x.map(truncate(2, _)).mkString(",")
    case ShowDefault(_)   => "N/A"
  }
```

```
show(ShowStrings(List("Hello", "World")))
// res5: String = "Hello,World"
show(ShowDoubles(List(123.123456, 0.1234)))
// res6: String = "123.12,0.12"
```

# Solution 3: Enumeration (Dotty)

```
enum ShowValue {  
  case class ShowString(value: String)  
  case class ShowDouble(value: Double)  
  case class ShowStrings(value: List[String])  
  case class ShowDoubles(value: List[Double])  
  case class ShowDefault[A](value: A)  
}  
  
def show(value: ShowValue): String =  
  value match {  
    case ShowString(x)    => x  
    case ShowDouble(x)    => truncate(2, x)  
    case ShowStrings(x)   => x.mkString(",")  
    case ShowDoubles(x)   => x.map(truncate(2, _)).mkString(",")  
    case ShowDefault(_)   => "N/A"  
  }
```

```
show(ShowStrings(List("Hello", "World")))  
// res7: String = "Hello,World"  
show(ShowDoubles(List(123.123456, 0.1234)))  
// res8: String = "123.12,0.12"
```



# Solution 4: Interface

```
trait Show[A] {  
  def show(value: A): String  
}  
  
val showString: Show[String] = new Show[String] {  
  def show(value: String): String = value  
}  
  
val showDouble: Show[Double] = new Show[Double] {  
  def show(value: Double): String =  
    truncate(2, value)  
}  
  
def showList[A](showA: Show[A]): Show[List[A]] =  
  new Show[List[A]] {  
    def show(value: List[A]): String =  
      value.map(showA.show).mkString(",")  
  }  
  
def defaultShow[A]: Show[A] = new Show[A] {  
  def show(value: A): String = "N/A"  
}
```

```
showString.show("Hello")  
// res9: String = "Hello"  
  
showDouble.show(123.123456)  
// res10: String = "123.12"  
  
defaultShow.show(true)  
// res11: String = "N/A"  
  
showList(showString).show(List("Hello", "World"))  
// res12: String = "Hello,World"  
  
showList(showDouble).show(List(123.123456, 0.1234))  
// res13: String = "123.12,0.12"
```