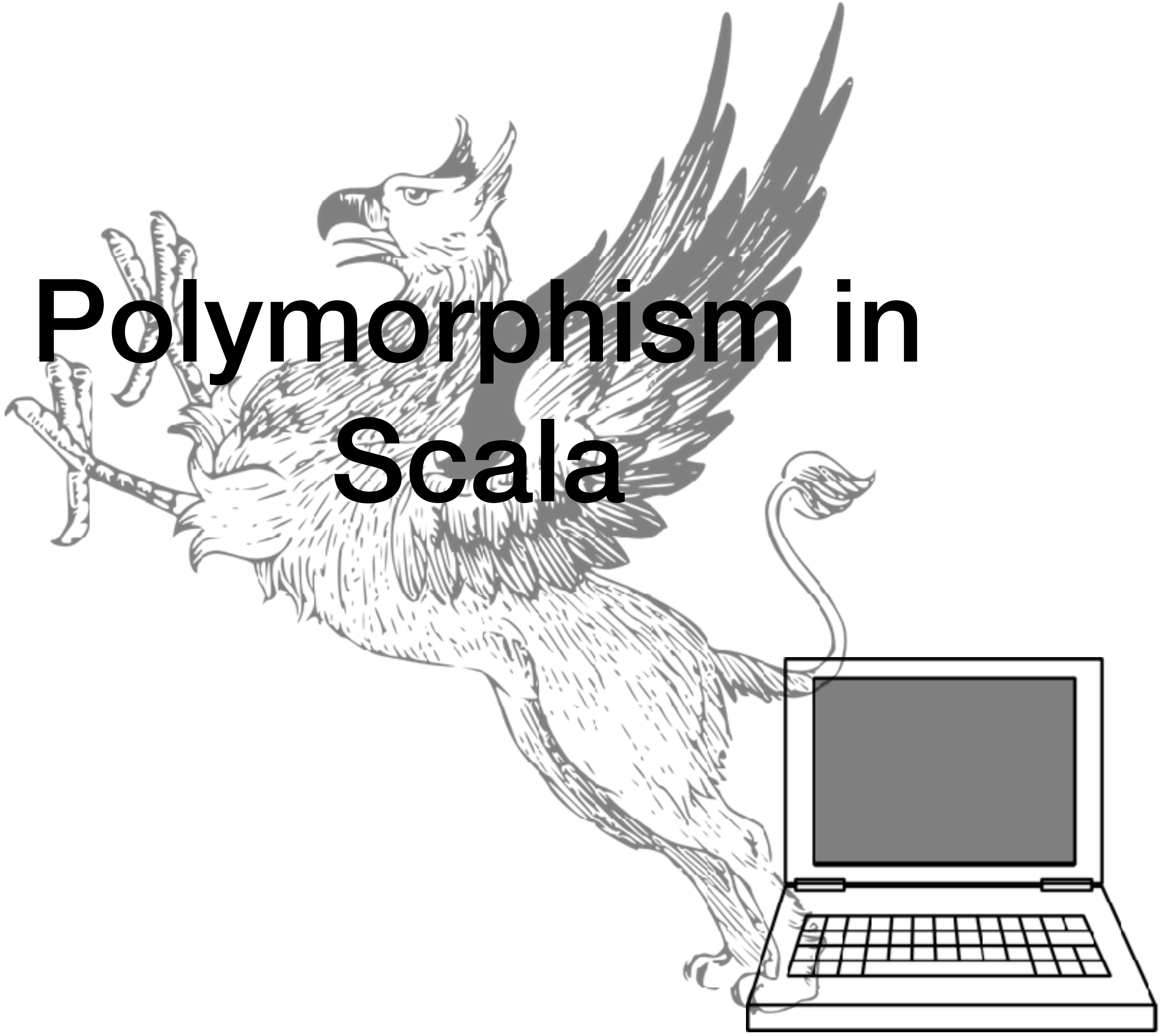
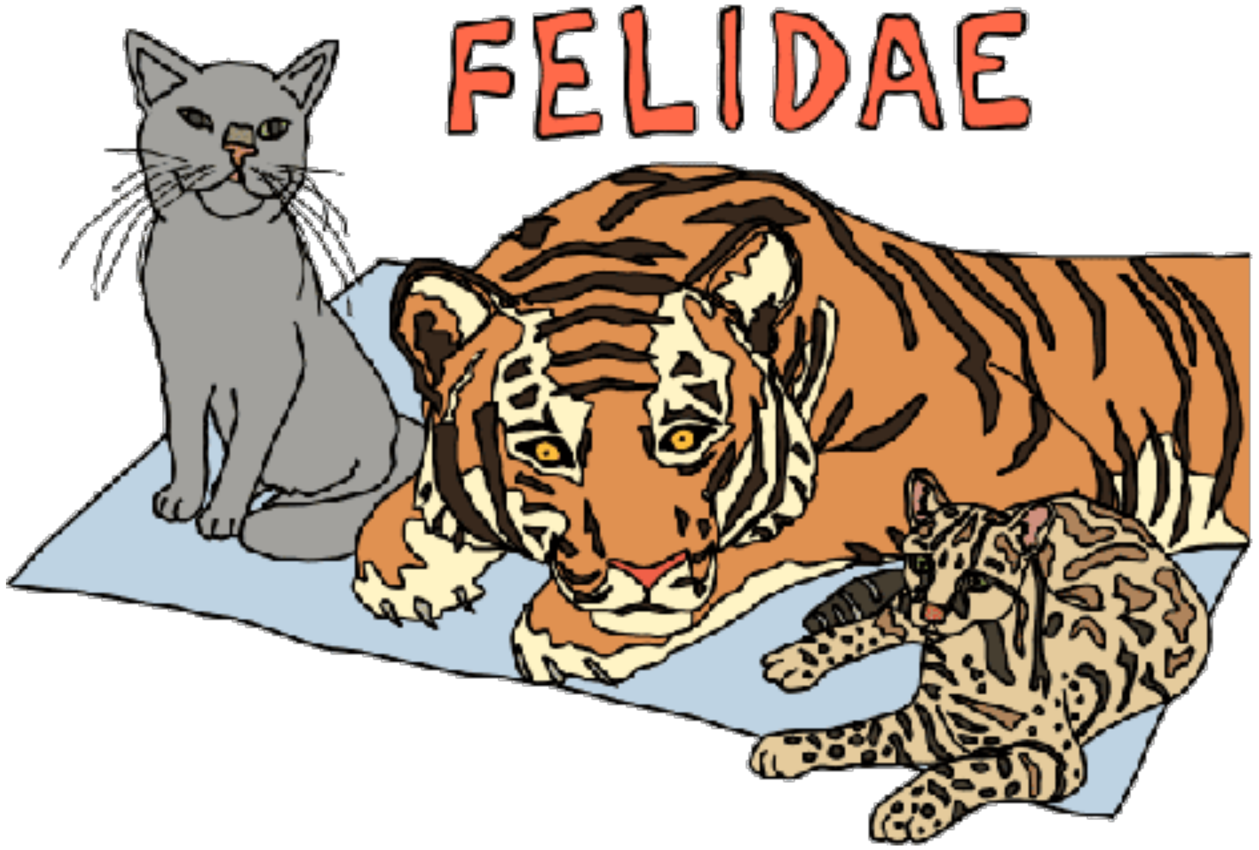
Polymorphism in Scala



What is polymorphism?

- Polymorphism means “many forms”
  - function can have arguments of “many types”
  - type can have instances of “many types”
- subtyping (OOP): instances of a subclass can be passed to a base class
- generics (FP): instances of class or function are created by type parametrization

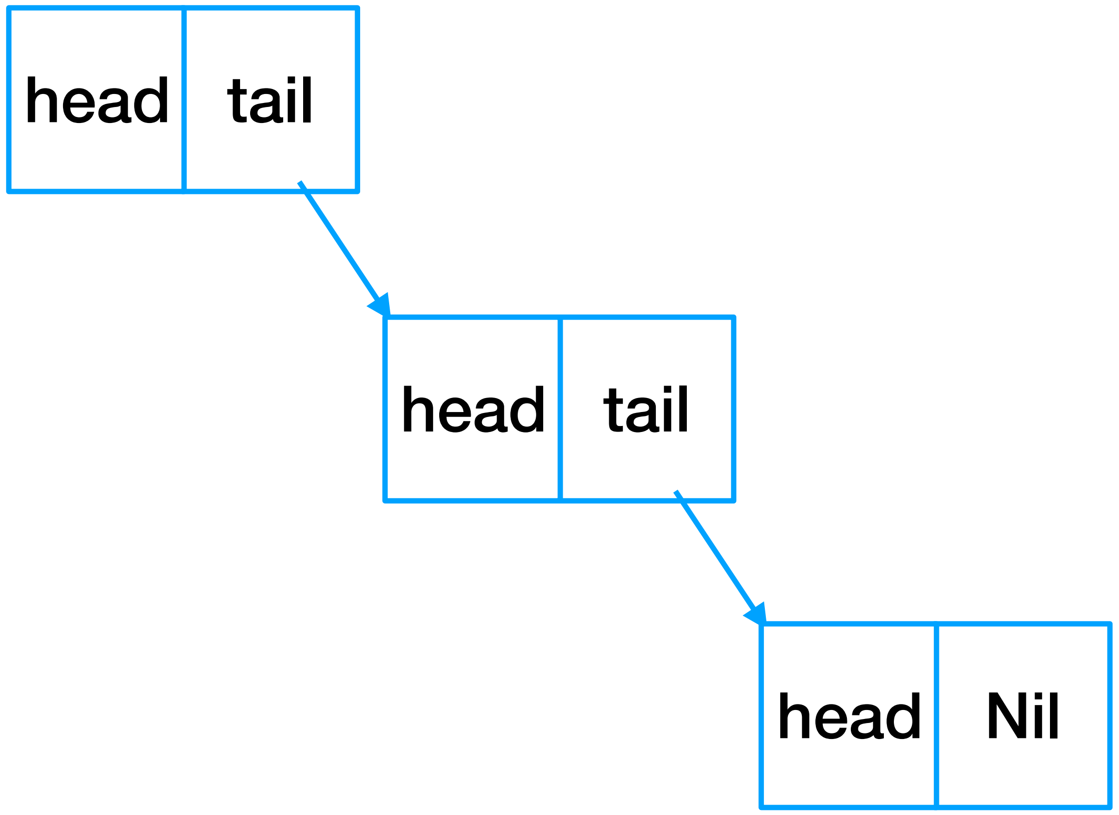


Generic Types

- Generic types are types (classes, traits) which take a type as a parameter
- Useful for collection classes
- Generic classes take a type as a parameter within square brackets []
- Subtyping of generic types is invariant,
  - Stack[A] is only a subtype of Stack[B] if and only if B = A

```
class Stack[A] {
  private var elements: List[A] = Nil
  def push(x: A) { elements = x :: elements }
  def peek: A = elements.head
  def pop(): A = {
    val currentTop = peek
    elements = elements.tail
    currentTop
  }
}
```

Demo Cons-list



```
polymorphism.List.scala
```

Polymorphic methods

- Like classes and traits, functions can have type parameters
- Ie, functions can be parameterized by type as well as value
- Type parameters are enclosed in square brackets, while value parameters are enclosed in parentheses
- Type can be implicit in call (type inference)
- Type erasure - only there at compile time, erased at runtime

```
def listOfDuplicates[A](x: A, length: Int): List[A] = {
  if (length < 1)
    Nil
  else
    x :: listOfDuplicates(x, length - 1)
}

scala> listOfDuplicates[String](“miaw!",4) //List[String] = List(miaw!, miaw!, miaw!, miaw!)

scala> listOfDuplicates(“moooh!",4) //List[String] = List(moooh!, moooh!, moooh!, moooh!)
```

Lab time!

polymorphism_07