LISA – *L*ISA *i*ncludes *S*ubclassing

LISA ist eine objektorientierte Programmiersprache, welche von Grund auf selbst entwickelt wurde. Sie ist statisch typisiert und syntaktisch an Scala und Java angelehnt. Der Fokus liegt auf dem innovativen Typensystem und nicht auf einer umfangreichen Programmiersprache, um gezielt das Konzept von *Subclassing* zu untersuchen.

Statisch typisierte Programmiersprachen und das Konzept des objektorientierten Programmierens konnten sich in vielen Bereichen der Softwareentwicklung etablieren. Entsprechend viele Sprachen und Modelle wurden entwickelt, um Software wiederverwendbar und wartungsfreundlich zu machen. Einige dieser Modelle, wie zum Beispiel *Generics* oder *Polymorphie*, wurden in weit verbreiteten Sprachen wie Java oder C# integriert. Zwar verbessern diese Modelle die Wiederverwendbarkeit von Softwarekomponenten, jedoch lösen sie nicht gewisse Schwierigkeiten, welche im Entwicklungsalltag auftreten. Dies betrifft insbesondere das Typensystem, weil dieses einerseits das Auftreten von gewissen Laufzeitausnahmen verhindert, andererseits teilweise das Kompilieren von sinnvollem Quellcode bezüglich der Wiederverwendbarkeit unterbindet. Als Beispiel soll die *equals* Methode der Klasse *Object* in Java betrachtet werden. Wird diese Methode in Klassen, welche immer *Object* als Supertyp haben, weiter spezialisiert, ist die Verwendung von typunsicheren Anweisungen, wie zum Beispiel *Casts*, unumgänglich.

Ähnliche Probleme können zwar mit *Generics* gelöst werden, jedoch stösst dieses Modell bei Typen, welche sich selbst als Methodenparameter erwarten, wie es bei *equals* von *Object* der Fall ist, an seine Grenzen. Eine direkte Erweiterung von *Generics* ist das *Subclassing*, welches das Parametrisieren vom eigenen Typ implizit umsetzt. LISA versucht das Modell von Subclassing zu realisieren. LISA versucht zusätzlich zum Subtyping-Vererbungsmodell das vom *Subclassing* in einer Sprache inklusive Compiler umzusetzen.

Links ist ein Beispiel einer Umsetzung in LISA gegeben, welches *Subclasses* und *Subtypes* verwendet und *Casts* unnötig macht.

```
class Animal {
  def mate(other: MyType): MyType {
    // Perform biological crossovers
  }
}

class Dog subclassOf Animal {
  def mate(other: MyType): MyType {
    // Specialising the argument 'other'
  }
}
```

```
class Cat subclassOf Animal {
  def mate(other: MyType): MyType {
    // Specialising the argument 'other'
  }
}

class Persian subtypeOf Cat {
  var needsGrooming: Bool;
  def mate(other: MyType): MyType {
    // Persian can mate other Cats
  }
}
```


Studenten: **Ruben Bär und Stefan Heinemann**
Betreuer: **Prof. Dr. Olivier Biberstein**