Concepts of Programming Languages

Object-oriented Languages

Prof. Dr. Guido Salvaneschi

Outline

- **Objects, classes, methods, fields**
- **Implementing OO languages**
- **Inheritance**
- **Implementing inheritance**
- **Summary**

So far: functional programs

```
val funDefs = Map(
    'f -> FunDef('x, App('g, Add('x, 3))),
    'g -> FunDef('y, Sub('y, 1)))

interp(App('f, 10), funDefs))
```

```
object P3 extends App {
  def factorial1(n: Int): Int = {
    if (n == 1) 1
    else n * factorial1(n - 1)
  }
  factorial1(6)
}
```

So far: functional programs

```
val box = Box(0)
box.value = 1 + box.value
box.value
```

with state

```
val a = Box(1)
def f(x: Int) = x + a.value
a.value = 2
println(f(5))
```

Today: Objects

Quiz: What are objects?

Objects

- Objects **encapsulate**:
  - data fields and
  - methods that operate on these fields
- These fields and methods belong together

Objects and Classes

- Objects **encapsulate**:
  - data fields and
  - methods that operate on these fields
- These fields and methods belong together

The methods and fields of an object are often defined in a class

- a class is typically used by multiple objects
- captures the commonalities of these objects (abstraction)

Prototype-based Languages

- Behaviour reuse (inheritance in OO terminology) is performed by **reusing** existing objects that serve as **prototypes** via **delegation**
- First language: Self
  - Self: the power of simplicity, Ungar and Smith, mid-80s
- JavaScript, Flash's ActionScript 1.0, Lua, Cecil, …

```
var father = {one: 1, two: 2};

// son.[[prototype]] = father
var son = Object.create( father );

son.three = 3;

son.one; // 1
son.two; // 2
son.three; // 3
```

Example Class

```
interp(whatDoesThisDo(iterations), store = new NoGCStore(storeSize))
```

```
class NoGCStore[Val](var maxSize: Int) extends Store[Val] {
    val memory = new scala.collection.mutable.ArraySeq[Val](maxSize)
    var freed = Set[Int]()

    var nextFreeAddr : Int = 0

    def malloc(stack: List[Map[Symbol, Int]], v: Val) = {
      if (!freed.isEmpty) {
        val next = freed.head
        freed -= next
        (next, this)
      }
      else {
        val x = nextFreeAddr
        if (x >= maxSize) sys.error("out of memory")
        nextFreeAddr += 1
        update(x, v)
        (x, this)
      }
    }

    def update(index: Int, v: Val) = {
```

Object-oriented languages

Quiz: What constructs does an object-oriented language consist of?

Object-oriented languages

Minimal OO language with classes:

- Class
  - fields
  - methods
  - at least one constructor

- Objects
  - instantiate exactly one class
  - bind concrete values to the fields of the class

State?

Quiz: Do we require (mutable) state?

State?

Quiz: Do we require (mutable) state?

No: objects may be immutable

```
class Point(x: Int, y: Int) {
  def setX(z: Int) = new Point(z, y)
  def setY(z: Int) = new Point(x, z)
}
```

Object-oriented languages

Things left out (for now):

- mutation of fields
- inheritance
- local variables in methods

Object-oriented languages

Quiz: What language constructs do we need for a minimal OO language?

Sketch the grammar…

Object-oriented languages

Quiz: What language constructs do we need for a minimal OO language?

```
<OO> ::= New <id> <OO>*
       | GetField <OO> <id>
       | Call <OO> <id> <OO>*
       | Id <id>
```

Class name

Object-oriented languages

Quiz: What language constructs do we need for a minimal OO language?

```
<OO>  ::= New <id> <OO>*
        | GetField <OO> <id>
        | Call <OO> <id> <OO>*
        | Id <id>
```

Receiver object

Object-oriented languages

Quiz: How do we represent object values?

```
<OO>  ::= New <id> <OO>*
        | GetField <OO> <id>
        | Call <OO> <id> <OO>*
        | Id <id>

<OOVal> ::= Object <id> <OOVal>*
```

Class name

Object-oriented languages

Quiz: How do we represent classes?

```
<OO> ::= New <id> <OO>*
       | GetField <OO> <id>
       | Call <OO> <id> <OO>*
       | Id <id>

<OOVal> ::= Object <id> <OOVal>*

<Class>  ::= Class <id>* <Method>*
<Method> ::= Method <id> <id>* <OO>
```

First-order vs first-class classes

First-order classes
- classes are predefined
- cannot be created dynamically

We consider first-order classes today

First-class classes
- classes can result from computations
- functions can be parameterized over class definitions

Object-oriented languages

Quiz: How do we encode first-order classes?

Object-oriented languages

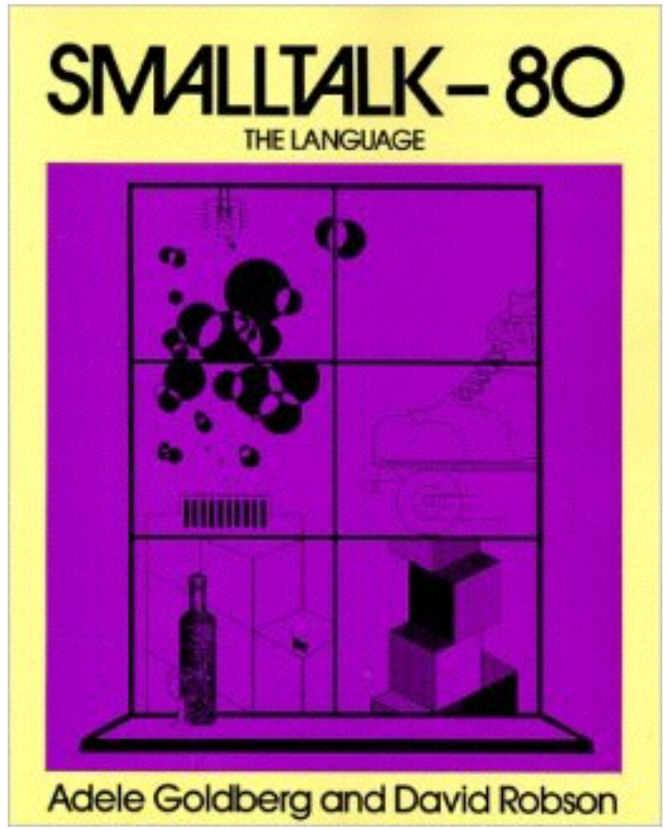
Quiz: How do we encode first-order classes?

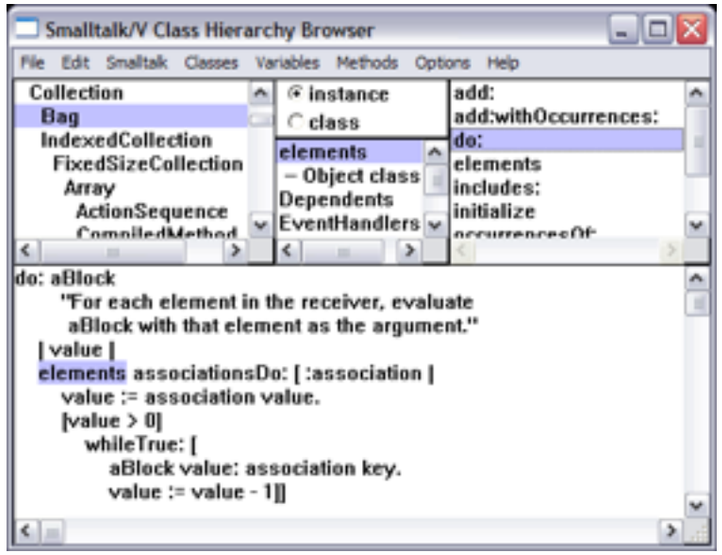
```
<OO> ::= New <id> <OO>*
       | GetField <OO> <id>
       | Call <OO> <id> <OO>*
       | Id <id>

<OOVal> ::= Object <id> <OOVal>*

<Class>  ::= Class <id>* <Method>*
<Method> ::= Method <id> <id>* <OO>

<ClassTable> = Table Map[<id> ~> <Class>]
```

Let’s implement the interpreter

Smalltalk

Rule 1. Everything is an object.
Rule 2. Every object is an instance of a class.
Rule 3. Every class has a superclass.
Rule 4. Everything happens by sending messages.
Rule 5. Method lookup follows the inheritance chain.





3 + 4 –→ 7 *"send '+ 4' to 3, yielding 7"*

20 factorial –→ 2432902008176640000 *"send factorial, yielding a big number"*

Outline

- Objects, classes, methods, fields
- Implementing OO languages
- Inheritance
- Implementing inheritance
- Summary

Object-oriented languages

Things we left out (so far):

- mutation of fields
- **inheritance**
- local variables in methods

Inheritance

Quiz: What is inheritance?

Inheritance

The methods and fields of an object are
often defined in a class

- a class is typically used by multiple objects
- captures the commonalities of these objects (abstraction)

Inheritance permits reuse of classes

- the class is used as a template for subclasses
- superclass can be specialized in multiple directions
- overriding of methods of superclass
- clients may be unaware of specialization

Inheritance

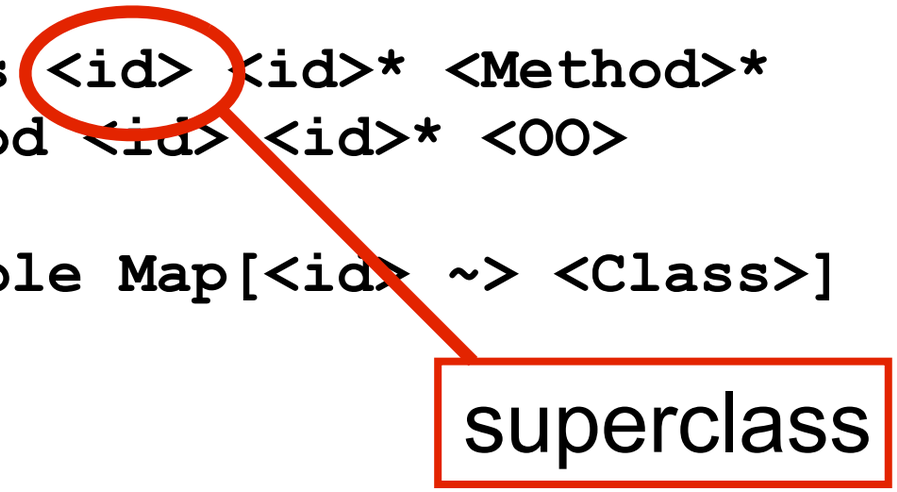
Quiz: What are the implications of inheritance on our language definition?

```
<OO> ::= New <id> <OO>*
       | GetField <OO> <id>
       | Call <OO> <id> <OO>*
       | Id <id>

<OOVal> ::= Object <id> <OOVal>*

<Class>  ::= Class <id>* <Method>*
<Method> ::= Method <id> <id>* <OO>
```

OO with inheritance

Quiz: How does the syntax change?

```
<OO> ::= New <id> <OO>*
       | GetField <OO> <id>
       | Call <OO> <id> <OO>*
       | Id <id>

<OOVal> ::= Object <id> <OOVal>*

<Class>  ::= Class <id> <id>* <Method>*
<Method> ::= Method <id> <id>* <OO>

<ClassTable> = Table Map[<id> ~> <Class>]
```



OO with inheritance

Quiz: How does the interpreter change?

Need to lookup fields and methods in superclass

```
class Point(x: Int, y: Int) {
  def setX(z: Int) = new Point(z, y)
  def setY(z: Int) = new Point(x, z)
}

class ZeroPoint(x: Int) extends Point(x, 0) {
  def setY(z: Int) = this
}

new ZeroPoint(15).setX(-15)
```

Let’s adapt the interpreter

Outline

- Objects, classes, methods, fields
- Implementing OO languages
- Inheritance
- Implementing inheritance
- Summary

Objects and Classes

- Objects encapsulate:
  - data fields and
  - methods that operate on these fields
- These fields and methods belong together

The methods and fields of an object are
often defined in a class

- a class is typically used by multiple objects
- captures the commonalities of these objects (abstraction)

Inheritance permits **reuse** of classes

Further topics

- OO versus functional programming
  - they can emulate each other

- local variables in methods
  - can coexist in the environment

- stateful OO: mutable fields
  - homework assignment