Name Resolution

Variables

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

Variable Definitions

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

Variable Usages

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

Variable Usages

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

What is the rule?
When can I use a variable?
What will its value be?

Lexical Scope and Lexical Binding

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

- Elements in the language (classes, methods, literal objects, closures…) define lexical scopes
- Variables defined in a scope are visible **within** that scope, but not outside
- Scopes can contain other scopes
- Inner scopes can access more things

Lexical Scope Variable Definition Example

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

- `Person`'s scope defines:

  `nextFreeChildName`

- `makeChild`'s scope defines:

  `prefix`

- the anonymous object's scope defines:

  `id`

- `initialize` and `getId` do not define variables

Lexical Scope Variable Reachability Example

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```

- In `makeChild` we can access:

  `nextFreeChildName`
  `prefix`

- In the anonymous object:

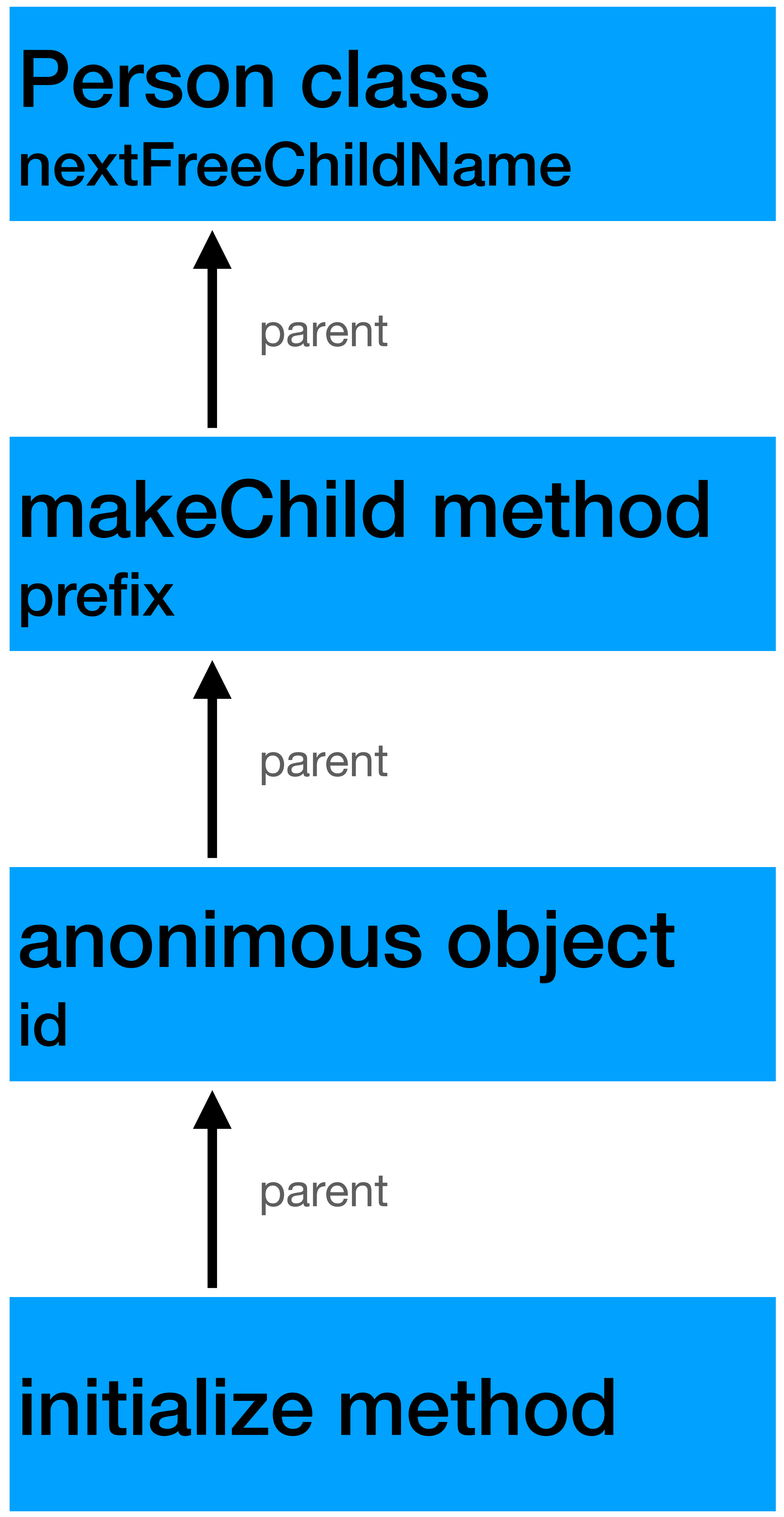
  `id` + all above

- In `initialize` and `getId`

  all above

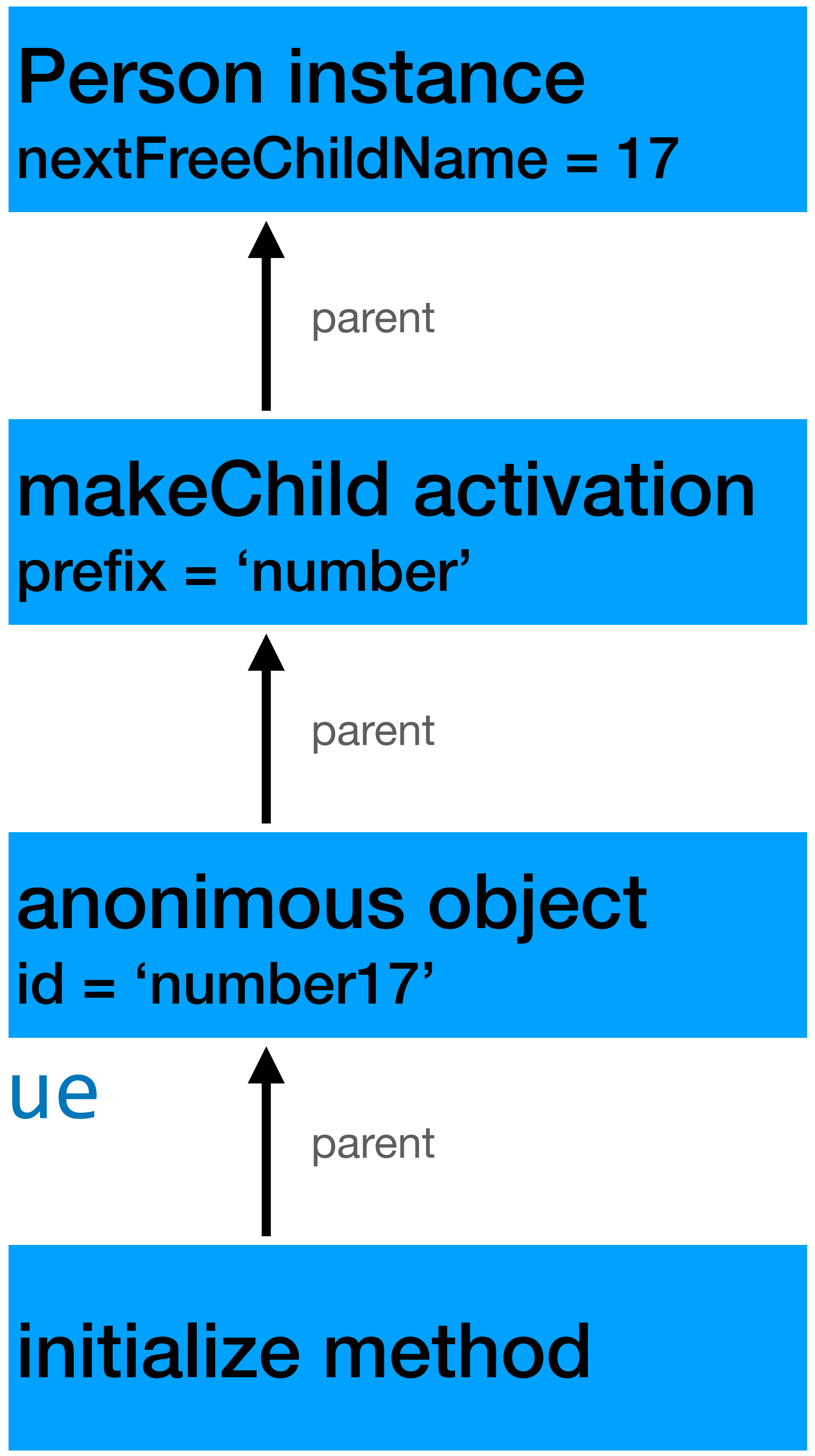
Lexical Scope As Chain Of Responsibility

```
class Person {
  var nextFreeChildName = 1;
  method makeChild(prefix) {
    return object {
      var id;
      method initialize(){
        id = prefix + nextFreeChildName.toString();
        nextFreeChildName += 1;
      }
      method getId() = id
    }
  }
}
```



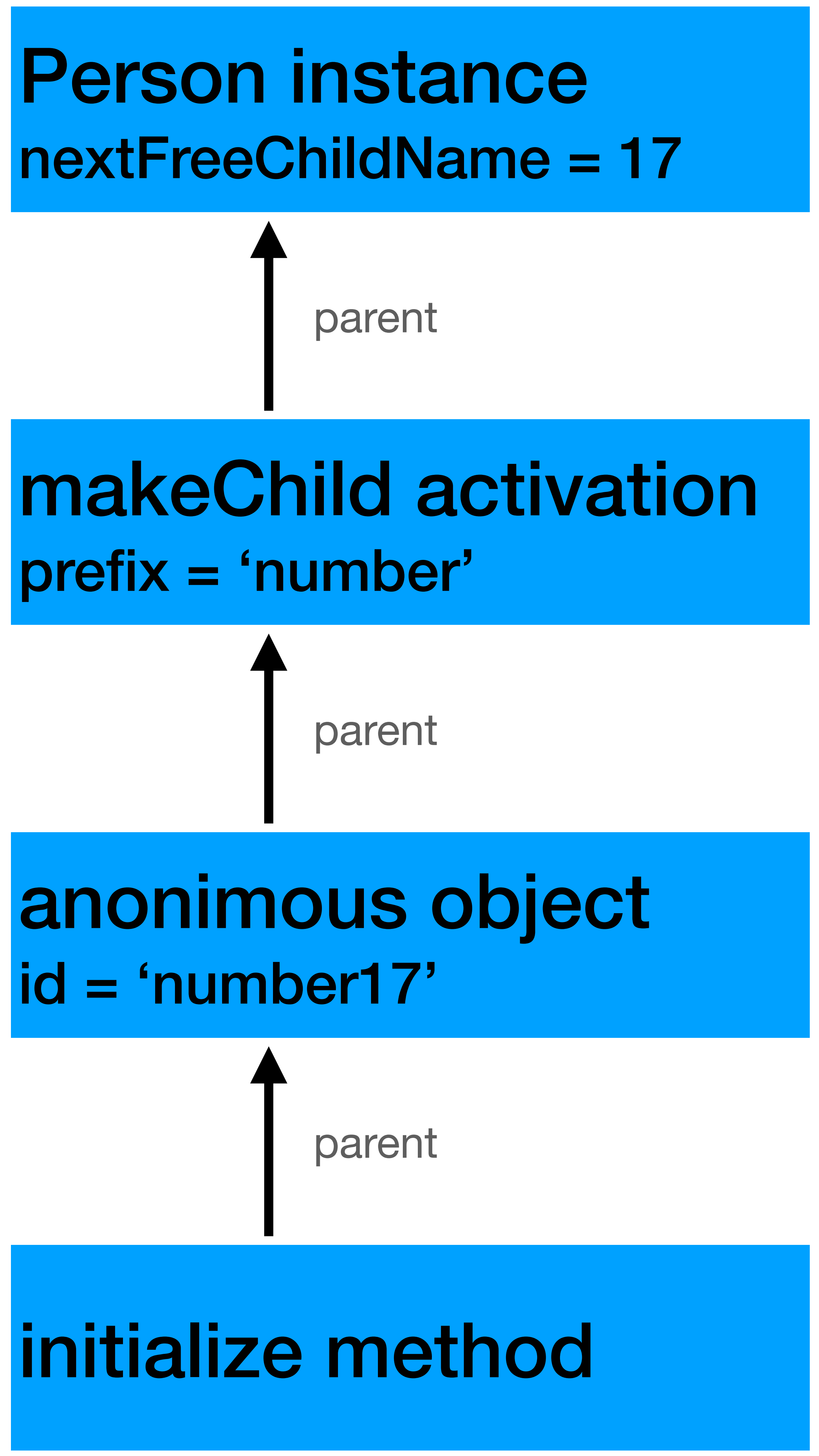
Implementing Lexical Scope Chain

```
Scope >> scopeDefining: name
  values at: name ifPresent: [:elem | ^ self ].
  ^ parentScope readVariableNamed: name

Interpreter >> readVariableNamed: name
  (self scopeDefining: name)
     read: name

Interpreter >> writeVariableNamed: name value: aValue
  (self scopeDefining: name)
     write: name value: aValue
```



Lexical closures

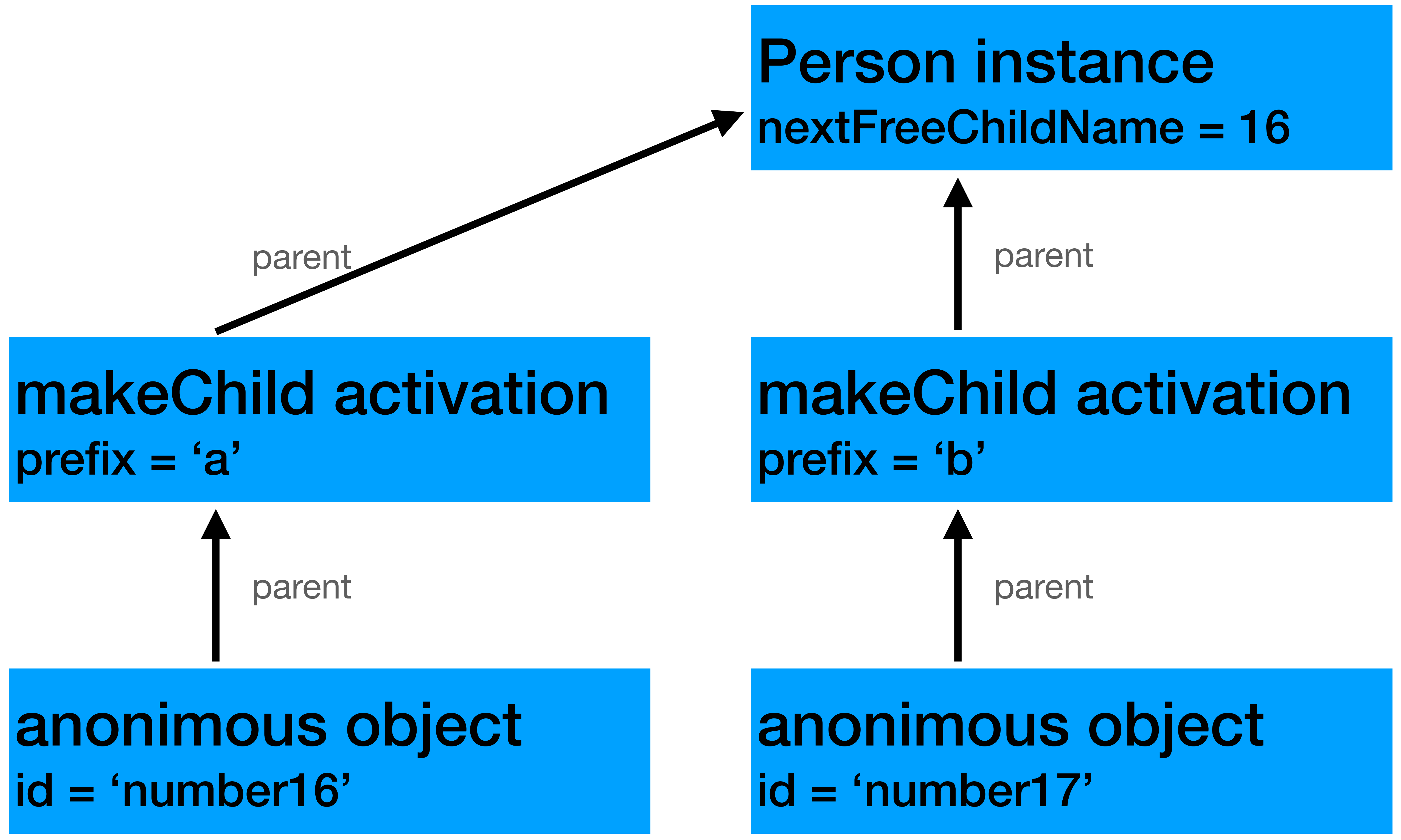
- Some language elements implement closures
- Closures *capture* its parent (and the variables in it)
- Every time a new closure executes, it will capture something new
- In Wollok, *anonymous objects* and *closure objects* implement closures
- For example, every time we call makeChild, a new anonymous object is created, capturing the activation of makeChild.



Lexical closures Example

- For example, every time we call makeChild, a new anonymous object is created, capturing the activation of makeChild.

```
var person = new Person();
person.makeChild(“a”);
person.makeChild(“b”);
```



Conclusion

- Elements in the language (classes, methods, literal objects, closures…) define lexical scopes
- Variables defined in a scope are visible **within** that scope, but not outside
- Scopes can contain other scopes
- Inner scopes can access more things

- Some elements, named closures, capture their parent scopes and can read and write on them