Super Methods

Calling up the Evolutionary Tree

Jim Leek, Tom Epperly, & Gary Kumfert

Center for Applied Scientific Computing

January 27, 2005

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.
UCRL-PRES-209186

What is a Super Method?

- Look! Up in the sky! It's a bird! It's a ….
- OOP languages provide a way of calling overwritten super class methods.
- 'Super Class Method' = 'Super Method.'
- In Java, the syntax is:
  - super.foo()

Motivation

- Super methods allow code reuse.
- Provides a way of accessing parent data.
- It was sitting around half completed.

How is it done?

- Supers may only be called from the Impls.
- Every binding is different.
- The IOR already has super tables and data.
- An IOR function gives us the vtable for just the first super class.
- We call on that vtable with the current object.
- We only generate OVERWRITTEN methods.

How is it done? (example)

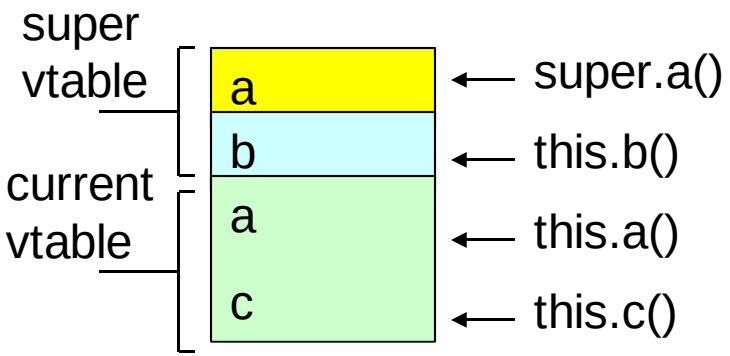
Class C inherits from B, which inherits from A.

A defines method a, B defines method b.

C defines method c, and overwrites method a.

Class Hierarchy

Virtual Function Table



How to call supers in C

- If we want to call super.a() from C_Impl.c (as shown in our example) we would:
- Prefix ‘super_’ to the super method:

```
char* impl_C_a(C self) {
/* DO-NOT-DELETE splicer.begin(C.a) */
  char* ret = super_a(self);
  return ret;
/* DO-NOT-DELETE splicer.end(C.a) */
```

How to call supers in Cxx/UCxx

- Prefix ‘super.’ to the super method:

```
::std::string C_impl::a () throw () {
  // DO-NOT-DELETE splicer.begin(C.a)
  ::std::string ret = super.a();
  return ret;
  // DO-NOT-DELETE splicer.end(C.a)
}
```

How to call supers in Fortran 77

Fully qualify the super method with: class_super_name_f()

```
 subroutine C_a_fi(self, retval)
       implicit none
C       in C self
       integer*8 self
C       out string retval
       character*(*) retval

C       DO-NOT-DELETE splicer.begin(C.a)
       call C_super_a_f(self, retval)
C       DO-NOT-DELETE splicer.end(C.a)
end
```

How to call supers in Fortran 90

- Prefix ‘super_’ to the super method:

```
recursive subroutine C_a_mi(self, retval)
  use C
  use C_impl
  implicit none
  type(C_t) :: self ! in
  character (len=*) :: retval ! out

! DO-NOT-DELETE splicer.begin(C.a)
  call super_a(self, retval)
! DO-NOT-DELETE splicer.end(C.a)
end subroutine C_a_mi
```

How to call supers in Java

- Prefix ‘super_’ to the super method:

```
public java.lang.String a_Impl ()
 {
   // DO-NOT-DELETE splicer.begin(C.a)
   java.lang.String ret = super_a();
   return ret;
   // DO-NOT-DELETE splicer.end(C.a)
 }
```

Conclusion

- Supers should ease working with Babel when Impls contain data.
- Make Babel more OOP!
- Unfortunately, Python does not yet work. (delayed)