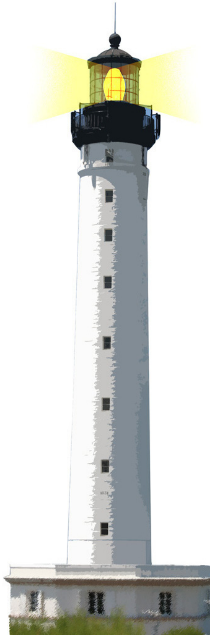
Advanced Object-Oriented Design

Private Methods in Java



http://www.pharo.org

Private Methods

Are private methods inherited?

```
class A {
  public void m() { this.p(); }
  private void p() { println("A.p()"); }
}
class B extends A {
  private void p() { println("B.p()"); }
}
```

Which is called? A.p() or B.p()?

```
A b = new B();
b.m();
```

Private Methods

Are private methods inherited?

```
class A {
  public void m() { this.p(); }
  private void p() { println("A.p()"); }
}
class B extends A {
  private void p() { println("B.p()"); }
}
```

Which is called? A.p() or B.p()?

```
A b = new B();
b.m();
```

```
A.p()
```

Because private methods are statically bound in Java

Private Methods in C++

- In C++ private can also be virtual

Private Methods in Ruby

In Ruby private methods are dynamically bound :)

Private Methods in Ruby

```
class C
  def fooAccessingX; x; end
  private
  def x; return 1; end
end
class D < C
  public
  def x; return 2; end
end
```

Results:

```
C.new.fooAccessingX ==> 1
D.new.fooAccessingX ==> 2
```

- The private method `x` is publicly redefined in a subclass
- Template superclass senders invoke the overriden method `x`

Private methods are accessible internally

Different ways to invoke methods:

- self.x uses the "external" interface while x the internal one

```
class C
  def fooSendingSelfX ; self.x end
  private
  def x; return 1; end
end
class D < C
  public
  def x; return 2; end
end
```

Results:

```
C.new.fooSendingSelfX ==> failed
D.new.fooSendingSelfX ==> 2
```

Object arguments uses the "external" interface

```
class C
  def zork(arg) ; return arg.x ; end
  def fooSendingSelfX ; self.x end
  def fooAccessingX; x; end
  private
  def x; return 1; end
end
class D < C
  public
  def x; return 2; end
end
```

```
C.new.zork(C.new) ==> failed
C.new.zork(D.new) ==> 2
```

Conclusion

Pay attention when using a private method

- You do not create a hook creation
    - Remember sending a message is a plan for reuse
- You break the extender interface (See Dual Interface Lecture)

A course by

S. Ducasse, G. Polito, and Pablo Tesone