Why OOP isn't always "the right thing"

Andrew Wright

Modularity

- abstract datatypes
  - BNF-like type declaration & set of operations
  - emphasizes organization by behavior
- objects
  - encapsulated data accessed by methods
  - emphasizes organization by kind of data

NEC

Designing with datatypes

```
datatype exp =
    Var    of name
  | Const  of value
  | Fun    of name list * exp
  | Ap     of exp * exp list
  | Obj    of exp list
```

```
typecheck = fn(e, env) =>
  case e of
    Var(x) => ...
    Const(v) => ...
    Fun(args, body) =>...
    Ap(fun, arg) => ...
    Obj(methods) => ...
```

```
normalize = fn(e, env) =>
  case e of
    Var(x) => ...
    Const(v) => ...
    Fun(args, body) =>...
    Ap(fun, arg) => ..
    Obj(methods) => ...
```

```
optimize = fn(e, env) =>
  case e of
    Var(x) => ...
    Const(v) => ...
    Fun(args, body) =>...
    Ap(fun, arg) => ...
    Obj(methods) => ...
```


NEC

Designing with Objects

```
abstract class Glyph
  method draw();
  method erase();
  method move(int,int);
  method rotate(int);
```

```
class Rectangle : Glyph
  draw() { }
  erase() { }
  move(int x, int y) { }
  rotate(int degrees) { }
```

```
class Circle : Glyph
  draw() { }
  erase() { }
  move(int x, int y) { }
  rotate(int degrees) { }
```

```
class Text : Glyph
  draw() { }
  erase() { }
  move(int x, int y) { }
  rotate(int degrees) { }
```

NEC

Designing appropriately

- use datatypes when enhancements will likely be new behaviors
- use objects when enhancements will likely be new kinds of data

NEC

Implications for languages

- programming languages should support
  - datatypes and objects
  - procedural and object-oriented programming

But Most Don't.

Support for objects and datatypes

	Datatypes	Objects
C	X	X
Scheme	can build	can build
ML	yes	X
Ada	yes	X
old C++	X	yes
new C++	awkward	yes
Java	awkward	yes

NEC

Faking datatypes in Java

- **make an abstract class with NO methods for the datatype:**

```
abstract class exp { }
```


NEC

Faking datatypes in Java

- **make concrete classes for each variant:**

```
class Var    :: exp {  name : string; };
class Const  :: exp { value: int; };
class Fun    :: exp { names: string list;   body : exp; };
class Ap     :: exp { fun : exp;   args : exp list; };
```

NEC

Faking datatypes in Java

- **use** instanceof **to perform case dispatching:**

```
typecheck( e : exp, env : environ ) {
   if(       e instanceof Var ) { Var v = (Var) e; … }
   else if( e instanceof Const ) { Const c = (Const) e; … }
   else if( e instanceof Fun ) { Fun f = (Fun) e; … }
   else if( e instanceof Ap ) { Ap a = (Ap) e; …}
```

May 17 1996 NEC Research Institute

NEC

Conclusions

- **a programming language that supported both abstract datatypes and objects well would facilitate more modular programming**
- **need more programming languages research**

May 17 1996 NEC Research Institute