Object Oriented Programming

1

OOP(S)

2

One-Slide Summary

- There are several (overlapping) kinds of polymorphism:
  - subtype
  - ad-hoc
  - parametric
- I like generic programming

3

Metaphysics

- Why are we here?
- What is 'polymorphism' anyway?

- How does this relate to car insurance?

4

Motivation (1)

- Typical example—containers:

```
class IntList {
   //...
   public int get(int i) {
      //...
   }
}
class StringList {
   // ...
   public String get(int i) {
      //...
   }
}
```

5

Motivation (2)

- Wouldn't it be nice if we could not-rewrite our code for every element type?
- Typical Example Continued:

```
class ObjectList {
   //...
   public Object get(int i) {
      //...
   }
}
```

6

Motivation (3)

```
class ObjectList {
   //...
   public Object get(int i) {
      //...
   }
}

PLResearcher wes = ...;
myList.add(wes)

Researcher x = (Researcher)myList.get(0);
// Does this work?
```

7

Polymorphism

```
PLResearcher wes = ...;
myList.add(wes)

Researcher x = (Researcher)myList.get(0);
```

- This is polymorphism; our List class now works for different element types
- More importantly, we only needed to code it once

8

Polymorphism (2)

- Several types:
  - Subtype polymorphism (as featured just now)
  - Ad-hoc polymorphism (similar to overloaded operators)
  - Parametric polymorphism (same code works for all types)

9

Guided Questions

- Are these kinds of polymorphism mutually exclusive?
- Did our List example 'work for all types?'
- Is the Godfather Object the best way to implement parametric polymorphism?

10

Problems with Object

- Before Object, we could do this:

```
StudentListList teams; // list containing Lists

StudentList team_a;
StudentList team_b;
teams.add(team_a) ; teams.add(team_b);
//...

// Get the second student from the first team
Student two_of_one = teams.get(0).get(1)
```

11

Problems with Object (2)

- Now, we need to cast:

```
List teams; // list containing StudentLists

List team_a;
List team_b;
teams.add(team_a) ; teams.add(team_b);
//...

// Get the second student from the first team
Student two_of_one =
    (Student) ((List)teams.get(0)).get(1)
```

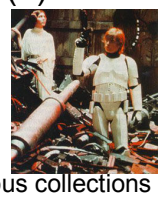
12

The Bad Place (3)

```
case teams.get(0) of
 slist : List =>
  case slist.get(1) of
   x : Student => x.blah()
  esac;
esac
```

13

The Bad Place (4)



- Casting is error prone
- Might cause runtime errors
- No way to ensure homogenous collections
- So Java < 5…

14

Parameterized Types

- The Typical Example, continued:

```
class List<T> {
   //...
   public T get(int i) {
      //...
   }
}

List<Dog> myList;
Dog milo = ...;
myList.add(milo)

Dog x = myList.get(0);
```

15

The Cunning Plan

- Let's add templates to COOL
  (since we have nothing better to do)
- We'll keep things simple:
  - one type parameter per class
  - can't do '`new T`' or '`case e of x : T`'
    (i.e. we really just want to rewrite the casts)

16

Adding PT to COOL (2)

- While we're here, let's redefine types altogether:

$$T ::= C \mid P\langle t \rangle \mid t$$

C : Normal Type (e.g. StudentList)
P<t> : Parameterized Type (e.g. List<T>)
t : A type parameter
(e.g. T within List<T>{ … })

17

Adding PT to COOL (3)

- Previously our typing judgments had the form:

$$\mathbf{O, M, C} \vdash \mathbf{e : T}$$

- Instead of just C, we need C | P<T>
- We'll call it **W**

18

Adding PT to COOL (4)

- We now define some new judgments to check types:

$$\frac{}{W \vdash C : \text{type}}$$

$$\frac{}{W{=}P\langle t\rangle \vdash t : \text{type}}$$

$$\frac{W \vdash T : \text{type}}{W \vdash P\langle T\rangle : \text{type}}$$

Let **W ⊢ T : type** denote that T is a valid type within class definition W

19

Adding PT to COOL (5)

$$\frac{\begin{array}{c}O, M, W \vdash e_0 : P\langle T\rangle \\ O, M, W \vdash e_1 : T_1 \\ M(P\langle x\rangle, f) = (T_1', T_2') \\ T_1 \le T_1''\end{array}}{O, M \Leftrightarrow \mathcal{W} \vdash e_0.f(e_1) : T_2''}$$

- What's wrong with this picture?
  - T_1'' is the result of replacing x with T in T_1'
  - T_2'' is the result of replacing x with T in T_2'

20

Adding PT to COOL (6)

- Need to add power to ≤

$C_0 \le C_1$	if C_0 inherits C_1 { … }
$C \le P\langle T\rangle$	if C inherits P<T> { … }
$P\langle T\rangle \le C$	if P<t> inherits C { … }
$P\langle T\rangle \le R\langle T\rangle$	if P<t> inherits R<t> { … }

$t \le t \le \text{Object}$

- Conclusion: type parameters are only moved around, or used as Object

21

Guided Questions

- Sigh. Did we add any ‘new power’ to COOL?
- How might one implement the scheme we just described?

22

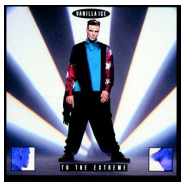
Postmortem

- Not all templates are created equal
- Java 1.5 Generics are very similar to what we just did with COOL
- What if we do allow ‘new’ and such?

23

Fun with C++ Templates

- C++ templates are Soooo 90s
- C++ templates are awesome
- Awesome = you can build Turing Machines



24

```
turing.cpp: In instantiation of 'Configuration<Q0,Pair<Blank,Pair<Bl
  TransitionFunction>':
turing.cpp:82:   instantiated from 'Configuration<Q1,Pair<Blank,Pair
turing.cpp:82:   instantiated from 'Configuration<Q0,Pair<Blank,Pair
turing.cpp:82:   instantiated from 'Configuration<Q1,Pair<Blank,Nil>
turing.cpp:82:   instantiated from 'Configuration<Q0,Pair<Blank,Nil>
turing.cpp:82:   instantiated from 'Configuration<Q1,Nil,Blank,Pair<
turing.cpp:82:   instantiated from 'Configuration<Q0,Nil,A,Pair<A,Pa
turing.cpp:163:   instantiated from here
turing.cpp:91: no type named 'halted_configuration' in 'struct Apply
  Pair<Blank,Pair<Blank,Nil> > >,Blank,Nil,TransitionFunction>'
```

Figure 1: Compiler errors from g++ 2.95.2. Reading the error messages $\vdash_M (q_1, \underline{\#}aa) \vdash_M (q_0, \#\underline{a}a) \vdash_M (q_1, \#\underline{\#}a) \vdash_M (q_0, \#\#\underline{a}) \vdash_M (q_1, \#\#\underline{\#}$

(http://osl.iu.edu/~tveldhui/papers/2003/turing.pdf)

25

Fun with C++ Templates (3)

- More explainable example:

```
template <int N>
struct fact {
  static const int value = N * fact<N - 1>::value;
};

template < >
struct fact<1> {
   static const int value = 1;
};

cout << fact<5>::value << endl; // result = ?
```

26

Fun with C++ Templates (4)

- Note that this looks almost… functional:

```
template <int N>
struct fact {
  static const int value = N * fact<N - 1>::value;
};

template < >
struct fact<1> {
   static const int value = 1;
};

cout << fact<5>::value << endl; // result = 120
```

27

Fun with C++ Templates (5)

- C++ templates are compile-time ad-hoc
- Let's try a less ethereal example:

```
template <typename EltType>
EltType & max(EltType & a, EltType & b) {
  return (a < b ? b : a);
}
```

- This works on any EltType that has `operator<` defined

28

Fun with C++ Templates (6)

- 'Compare' this with:

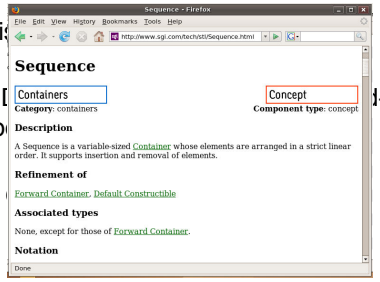
```
public interface LtComparable {
      public bool lessThan(LtComparable o);
}
//...

LtComparable max(LtComparable a, LtComparable b)
{
  return (a.lessThan(b) ? b : a);
}
```

- Note that multiple inheritance doesn't really help here

29

Fun with C++ Templates (7)

- This i
- TA's D -hoc p
- Most



30

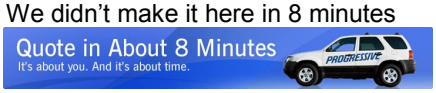
Shorter Version

- There are different kinds of polymorphism; not all are created equal
- We could add parametric-ish polymorphism to COOL with relative ease
- There is a reason why we disallowed dispatch on template parameter instances…

31

Postpostmortem

- We didn't make it here in 8 minutes



- WA4 Due Friday, 11:59pm
  - re-construct AST from cl-ast file
  - do some basic checks, e.g. no circular inheritance

32

Karma Points

```
NAME,INHERITS,CHILDCOUNT,CHILDREN,LINE = 0, 1, 2, 3, 4
lineno = 0
ast_tree_root = dict
f = open(sys.argv[1])
lines = [x.rstrip("\r\n") for x in f.readlines()]
f.close()

ast_tree_rt =
  dict( [(NAME, ""),
         (LINE, ""),
         (INHERITS, ""),
         (CHILDCOUNT, lines[0] + "\n"),
         (CHILDREN, build_classes(int(lines[0])))
        ])

sys.stdout.write(ast_tree_rt[CHILDREN][0][NAME])
```

33