Object Oriented Programming

One-Slide Summary

• There are several (overlapping) kinds of polymorphism:
  – subtype
  – ad-hoc
  – parametric

• I like generic programming
Metaphysics

• Why are we here?
• What is ‘polymorphism’ anyway?
• How does this relate to car insurance?

Motivation (1)

• Typical example—containers:

```java
public int get(int i) {
    //...
}
```

Motivation (2)

• Wouldn’t it be nice if we could not-rewrite our code for every element type?

• Typical Example Continued:

```java
public Object get(int i) {
    //...
}
```
```java
class ObjectList {
    //...
    public Object get(int i) {
        //...
    }
    //...
} //Does this work?

PLResearcher wes = ...;
myList.add(wes)
Researcher x = (Researcher)myList.get(0);
```

### Motivation (3)

### Polymorphism

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

### 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)
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?

Problems with Object

• Before Object, we could do this:

```java
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)
```

Problems with Object (2)

• Now, we need to cast:

```java
List teams; // list containing Student lists
List team_a;
List team_b;
team.add(team_a); teams.add(team_b);
//...
// Get the second student from the first team
(Student) ((List)teams.get(0)).get(1)
```
The Bad Place (3)

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

The Bad Place (4)

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

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);
```
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)

Adding PT to COOL (2)

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

\[ T ::= \text{C} \mid \text{P}<\text{t}> \mid \text{t} \]

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

Adding PT to COOL (3)

- Previously our typing judgments had the form:

\[ \text{O, M, C} :: e : \text{T} \]

- Instead of just \text{C}, we need \text{C} \mid \text{P}<\text{T}>
- We’ll call it \text{W}
Adding PT to COOL (4)

- We now define some new judgments to check types:

  \[ W \vdash C : \text{type} \]

  \[ W = P\langle t\rangle \vdash t : \text{type} \]

  \[ W \vdash T : \text{type} \]

  \[ W = P\langle T\rangle : \text{type} \]

Let \( W \vdash T : \text{type} \) denote that \( T \) is a valid type within class definition \( W \).

Adding PT to COOL (5)

\[
\begin{align*}
O, M, W \vdash e_0 : P\langle T\rangle \\
O, M, W \vdash e_1 : T_1 \\
M(P\langle x\rangle, t) = (T_1, T_2) \\
T_1 \leq T_1"
\end{align*}
\]

\[
O, M \Rightarrow W \vdash e_0.f(e_1) : T_2"
\]

- 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 \( T_2\).

Adding PT to COOL (6)

- Need to add power to \( \leq \)

  \[
  \begin{align*}
  C_0 \leq C_1 & \quad \text{if } C_0 \text{ inherits } C_1 \{ \ldots \} \\
  C \leq P\langle T\rangle & \quad \text{if } C \text{ inherits } P\langle T\rangle \{ \ldots \} \\
P\langle T\rangle \leq C & \quad \text{if } P\langle t\rangle \text{ inherits } C \{ \ldots \} \\
P\langle T\rangle \leq R\langle T\rangle & \quad \text{if } P\langle t\rangle \text{ inherits } R\langle t\rangle \{ \ldots \} \\
t \leq t \leq \text{Object} & \quad \text{if } t \text{ inherits } \text{Object}
  \end{align*}
  \]

- Conclusion: type parameters are only moved around, or used as \( \text{Object} \).
Guided Questions
• Sigh. Did we add any ‘new power’ to COOL?
• How might one implement the scheme we just described?

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?

Fun with C++ Templates
• C++ templates are Soooo 90s
• C++ templates are awesome
• Awesome = you can build Turing Machines
Fun with C++ Templates (3)

• More explainable example:

```cpp
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 = ?
```

Fun with C++ Templates (4)

• Note that this looks almost... functional:

```cpp
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
```
Fun with C++ Templates (5)

- C++ templates are compile-time ad-hoc

- Let’s try a less ethereal example:

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

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

Fun with C++ Templates (6)

- 'Compare' this with:

  ```cpp
  public interface LtComparable {
    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

Fun with C++ Templates (7)

- This is ‘generic programming’

- TA’s [ad-hoc programming]

- Most
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...

Postpostmortem

- We didn't make it here in 8 minutes
- WA4 Due Friday, 11:59pm
  – re-construct AST from c-ast file
  – do some basic checks, e.g. no circular inheritance

Karma Points

```python
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_root = dict([(NAME, ""),
                      [LINE, ""),
                      [INHERITS, ""),
                      [CHILDCOUNT, lines[0] + "\n"),
                      [CHILDREN, build_classes(int(lines[0]))])
sys.stdout.write(ast_tree_root[CHILDREN][0][NAME])
```