

## More Examples

### AST optimization:

- Partial class hierarchy for an AST (Abstract Syntax Tree):

```
//Arithmetic expressions
abstract class Expr { ... }

//Binary operators
class Binop extends Expr { ... }

//'+' operator
class Plus extends Binop { ... }

//Numeric constants
class NumConst extends Expr { ... }
```

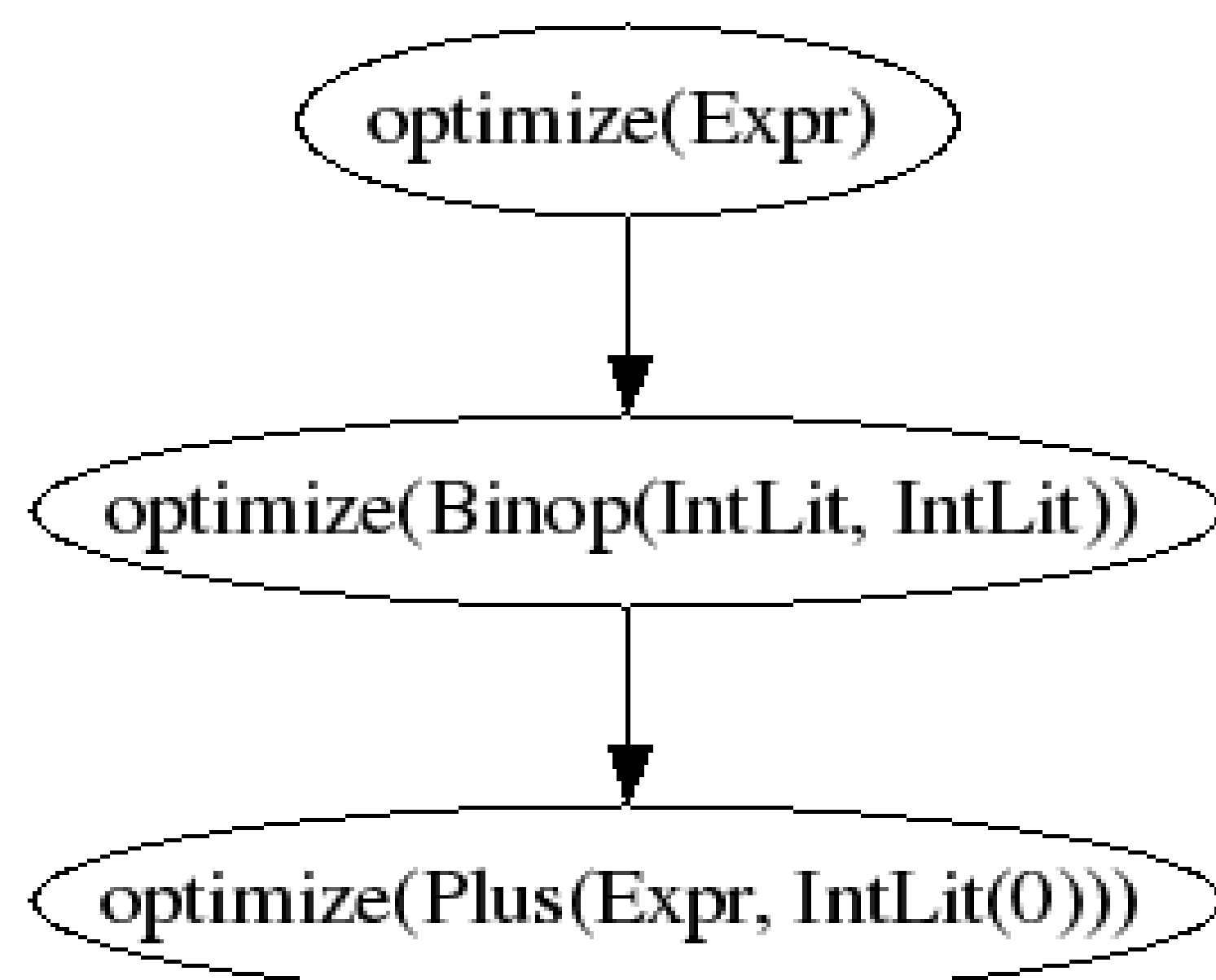
- Given the above classes, part of the code optimizer could be:

```
//do nothing by default
Expr optimize(Expr e) { return e; }

//Anything + 0 is itself
Expr optimize(Plus(Expr e, NumConst(0)))
{ return e; }

//Constant folding
| Expr optimize(Binop(NumConst c1,
                    NumConst c2) op)
{ return op.eval(c1, c2); }
```

- The “|” (normally the bitwise-or operator) is used to manually specify overriding
- Without it, there would be an error if both methods applied
- Hence, the overriding relationships here are:



### GUI Event Processing:

- Partial Widget class hierarchy

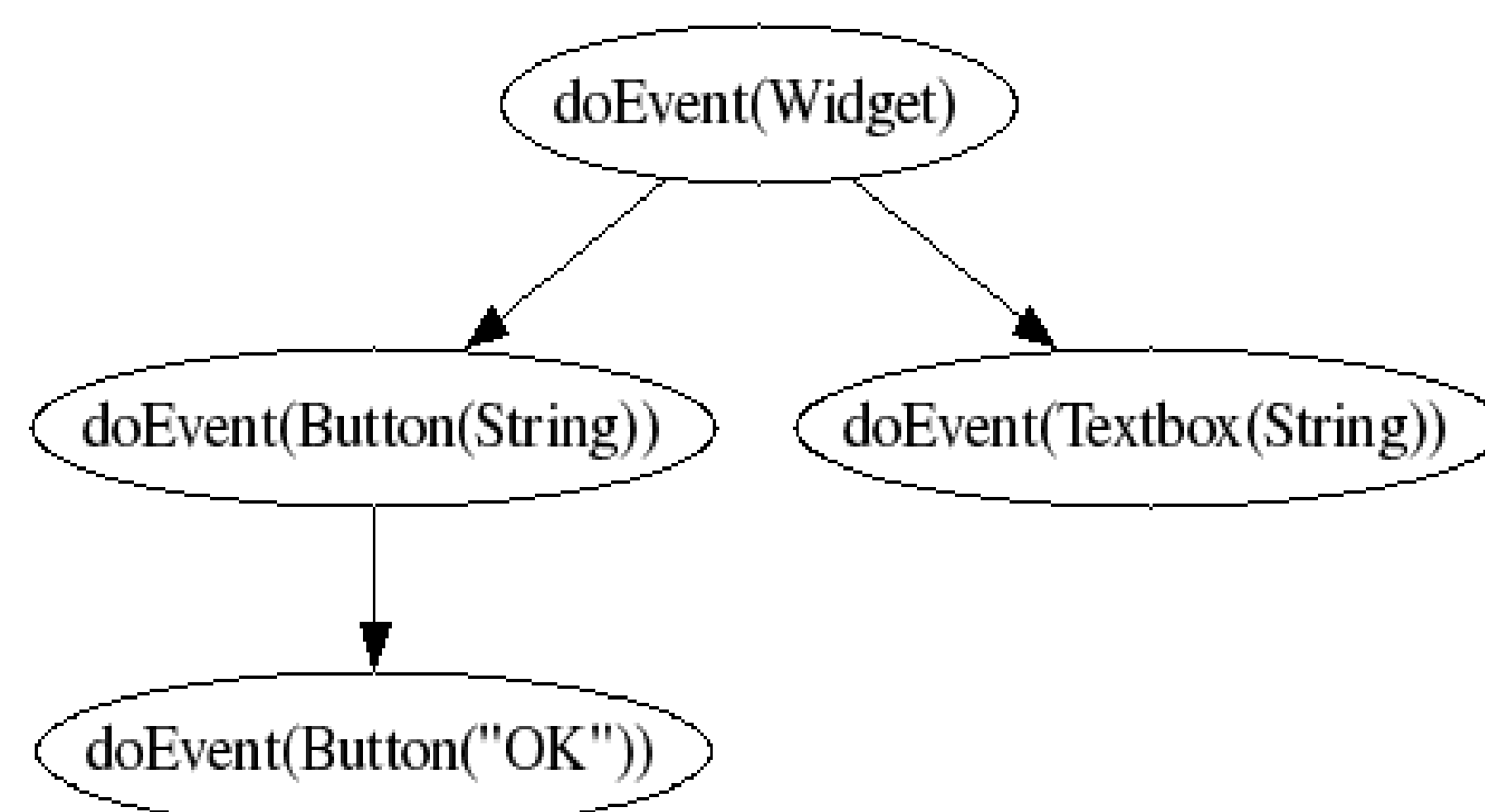
```
class Widget {
}

class Button extends Widget {
    public Button(public String name)
    {}
}

class Textbox extends Widget {
    public Textbox(public String text)
    {}
}
```

- Event processing can have a separate method for each widget or group of widgets:

```
void doEvent(Widget w) {}
void doEvent(Button("OK")) {
    //OK button clicked
}
void doEvent(Button(String name)) {
    //Process other buttons
    //(e.g. if there is an array of
    //buttons)
}
void doEvent(Textbox(String text)) {
    //Textbox event
}
```



## Errors

- Conditional dispatch introduces the possibility of new types of errors at run-time
  - **Ambiguity Error** (multiple methods apply to a call, and neither is most specific)
  - **No-such-method Error** (no method can be found for a call)
- Compiler prevents no-such-method errors by normally forcing all methods to override a **regular Java method**.
  - Programmer can circumvent this by labeling methods “**inc**” (to allow patterns to be used as preconditions)
- Compiler can find most ambiguity errors. It cannot find errors caused by:

- **Multiple (interface) inheritance**. E.g. for interfaces A and B:

```
void f(A a) {}
void f(B b) {}
```

- **Different destructors** on the same (or a related) class. E.g. for destructors d1 and d2::

```
void f(Point.d()) {}
void f(Point.d2()) {}
```

- **Destructors that modify the heap or aren't deterministic:**

```
destructor Point(int x, int y) {
    Random r = new Random();
    //Randomly return either 0 or 1
    //for each of x and y
    x = r.nextInt(2);
    y = r.nextInt(2);
}
```

## Implementation

- OOMatch has been implemented in Polyglot, a Java-to-Java compiler framework
- I.e. we translate OOMatch to Java
- Translation renames all methods, and creates dispatchers that select which method gets called
- Method calls call the dispatcher, instead of the original method
- Implementation only guaranteed to work for single-threaded programs. Concurrent programs are left for future work.