Pattern Enforcing Compiler (PEC) for Java: Use Cases for Multiple Dispatch

by

Howard C. Lovatt, Anthony M. Sloane, &
Dominic R. Verity

howard.lovatt@iee.org

pec.dev.java.net

Talk Outline

- PEC
- Multiple Dispatch in PEC
- Use Cases
  - Simplicity
  - Specialized Methods
  - Argument Symmetry
  - Increased Decoupling
  - Ultimate Extendability
  - Avoiding Type Specific Code
  - Instead of Pattern Matching
- Summary

PEC

- Pattern Enforcing Compiler (PEC)
- Extends normal type checking
- Allows a class to be marked as conforming to a pattern
- Syntax:

  class MySing implements Singleton

  - Analogous to

    int myInt

- **Allows you to write your own patterns**

Multiple Dispatch

```
public interface Shape extends MultipleDispatch {
  Shape intersect( Shape s );
}
public abstract class AbstractShape implements Shape {
  public final Shape intersect( Shape s ) { throw new AssertionError(); }
  public static final Shape intersect( Shape s1, Shape s2 ) {
    ... // General-purpose method
  }
}
public class Square extends AbstractShape {
  public static final Shape intersect( Square s1, Square s2 ) {
    ... // Square-Square method
  }
}
```

Simplicity

- Single Dispatch

```
class Square extends AbstractShape {
  public Shape
  intersect( Square s ) { ... }
}
```

- Multiple Dispatch

```
class Square extends AbstractShape {
  public final static Shape
  intersect( Square s1, Square s2 ) { ... }
}
```

- Test

```
Shape s = new Square();
s.intersect( s );
```

Specialized Methods

- Single Dispatch

```
class Square extends AbstractShape {
  public Shape intersect( Square s ) { ... }
}
```

- Multiple Dispatch

```
class Square extends AbstractShape {
  public final static Shape intersect( Square s1, Square s2 ) { ... }
}
```

Argument Symmetry

- Single Dispatch Circle

```
intersect( Circle c ) { ... }
intersect( Square s ) { ... }
// Can't add Square-Circle!
}
```

- Multiple Dispatch Circle

```
intersect( Circle s1, Circle s2 ) { ... }
intersect( Square s1, Circle s2 ) { ... }
intersect( Circle s1, Square s2 ) { ... }
```

- Test

```
Shape s = new Square();
s.intersect( s );
Shape c = new Circle();
c.intersect( c );
c.intersect( s );
s.intersect( c );
```

Increased Decoupling

- Single Dispatch (Visitor)

```
public interface ExpressionVisitor {
  void visitIntExp( IntExp e );
  void visitAddExp( AddExp e );
}
public interface Expression {
  void accept( ExpressionVisitor v );
  ...
}
```

- Multiple Dispatch

```
// Nothing

public interface Expression {
  ...
}
```

Increased Decoupling 2

- Single Dispatch (Visitor)

```
public class
IntExp implements Expression {
  public void
  accept( ExpressionVisitor v ) {
    v.visitIntExp( this );
  }
  ...
}
```

- Multiple Dispatch

```
public class
IntExp implements Expression {
  ...
}
public interface PrettyPrint
extends MultipleDispatch {
  void prettyPrint( Expression e );
}
```

Increased Decoupling 3

- Single Dispatch (Visitor)

```
public class PrettyPrint
implements ExpressionVisitor {
  public void
  visitIntExp( IntExp e ) ...
  public void
  visitAddExp( IntExp e ) {
    e.e1.accept( this );
    System.out.print( " + " );
    e.e2.accept( this );
  }
}
```

- Multiple Dispatch

```
public final class PrettyPrint
implements PrettyPrint {
  public final void
  prettyPrint( Expression e ) ...
  public final static void
  prettyPrint( PrettyPrint pp, IntExp e ) ...
  public final static void
  prettyPrint( PrettyPrint pp, AddExp e ) {
    pp.prettyPrint( e.e1 );
    System.out.print( " + " );
    pp.prettyPrint( e.e2 );
  }
}
```

Ultimate in Extendability

```
public class Triangle extends AbstractShape {
  public static final Shape intersect( Triangle t1, Triangle t2 ) {
    ... // Triangle-Triangle method
  }
  public static final Shape intersect( Triangle t, Square s ) {
    ... // Triangle-Square method
  }
  public static final Shape intersect( Square s, Triangle t ) {
    ... // Square-Triangle method
    // Could be intersect( t, s )
  }
}
```

Avoiding Casts

- Single Dispatch equals

```
public boolean equals( Object other ) {
  if( !(other instanceof Square ) return false;
  Square that = (Square)other;
  ...
}
```

- Multiple Dispatch equals

```
public static final boolean equals( Shape s1, Shape s2 ) { return false; }
public static final boolean equals( Square s1, Square s2 ) { ... }
```

Pattern Matching

- Single Dispatch

```
class Square implements Shape {
  public Shape intersect( Shape s ) {
    if ( s instanceof NullSize ) return s;
    ...
  }
}
class Circle ...
class NullSize implements Shape {
  public Shape intersect( Shape s ) { return this; }
}
```

Pattern Matching 2

- Single Dispatch (made up!)

```
class AbstractShape implements Shape {
  public final Shape intersect( Shape s ) {
    pattern ( this, s ) {
      match ( NullSize n, Shape s ) return n;
      match ( Shape s, NullSize n ) return n;
    }
    return intersectHasSize( s );
  }
  protected abstract Shape intersectHasSize( Shape s );
}
```

Pattern Matching 3

- Multiple Dispatch

```
public static final Shape intersect( Shape s, NullSize n ) { return n; }
public static final Shape intersect( NullSize n, Shape s ) { return n; }
```

Summary

- Outlines use cases for Multiple Dispatch
- Examples in our extended compiler we call a A Pattern Enforcing Compiler (PEC)
- Download (LGPL)
  - pec.dev.java.net