17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

ASPECT-ORIENTED PROGRAMMING

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH

PROJECT UPDATE

Please send me a project description (1-2 paragraphs) by the end of the week.

READING ASSIGNMENT NOV 9

Voelter, Markus, and Eelco Visser. "Product line engineering using domain-specific languages." *Software Product Line Conference (SPLC), 2011 15th International*. IEEE, 2011.

LEARNING GOALS

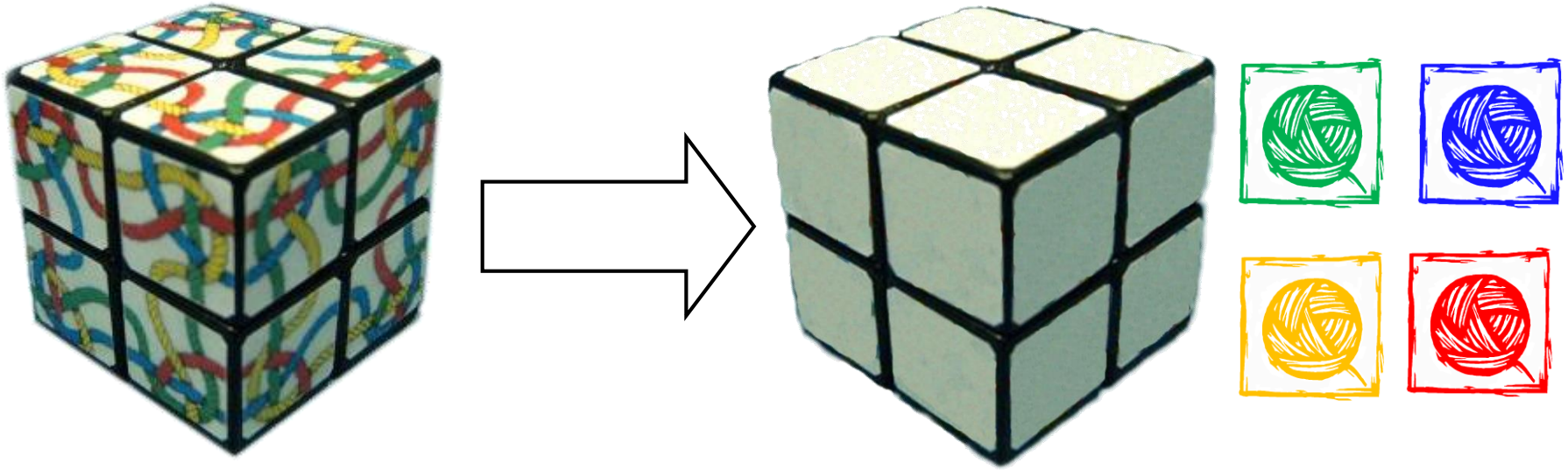
Understand key concepts of aspect-oriented programming

Implement variations with aspect-oriented programming

Understand modularity implications of aspect-oriented programming and mechanisms to mitigate issues

Compare aspect- and feature-oriented programming with other implementation strategies

Select a suitable implementation strategy for a given problem



INTER-TYPE DECLARATION

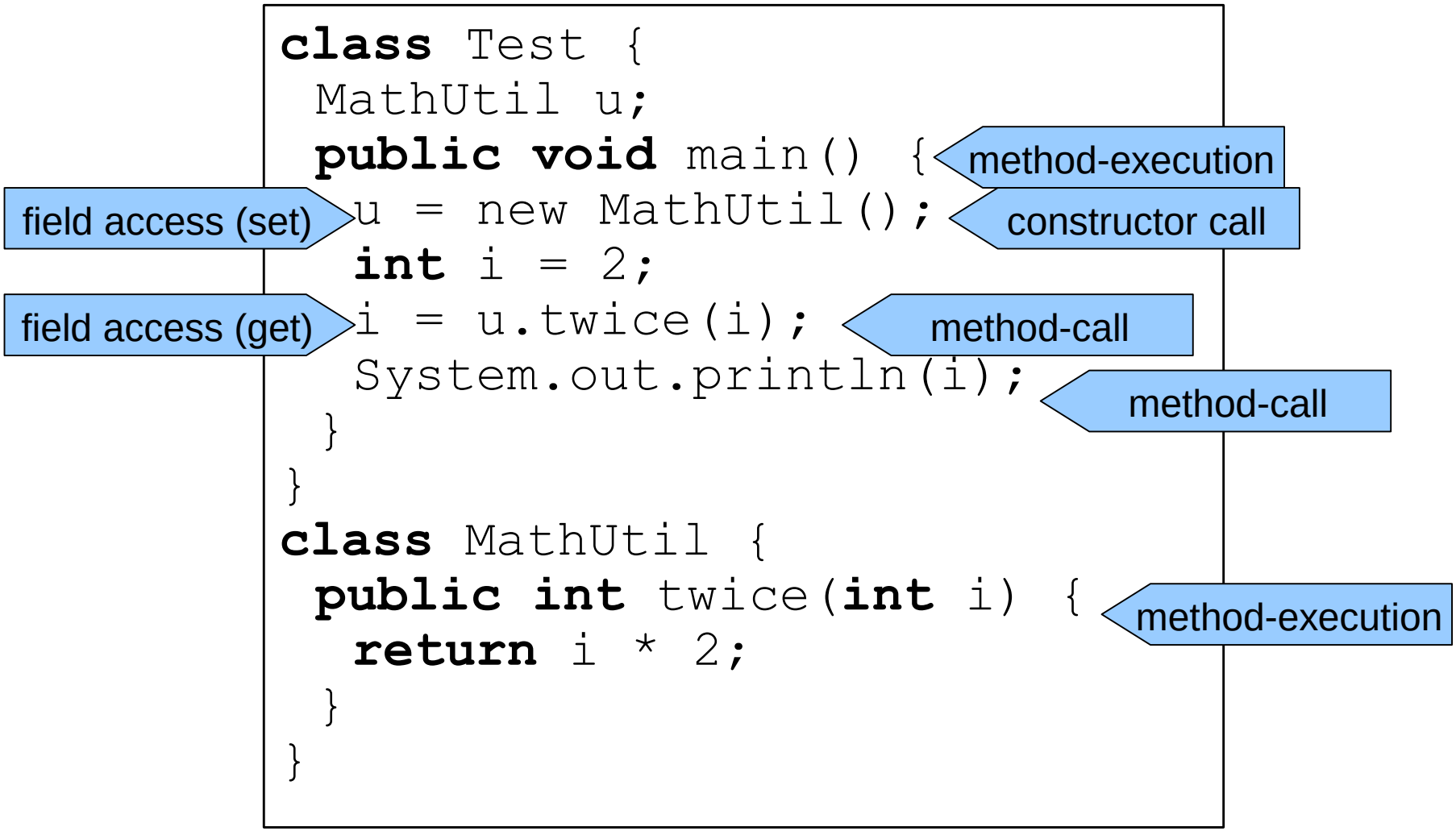
```
aspect Weighted {
        private int Edge.weight = 0;
        public void Edge.setWeight(int w) {
                weight = w;
        }
}
```

DYNAMIC EXTENSIONS

```
aspect Weighted {
    ...
    pointcut printExecution(Edge edge) :
        execution(void Edge.print()) && this(edge);

    after(Edge edge) : printExecution(edge) {
        System.out.print(' weight ' + edge.weight);
    }
}
```

QUANTIFICATION AND OBLIVIOUSNESS

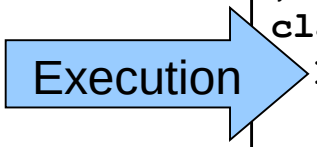
JOIN POINT MODEL



EXECUTION VS CALL

```
aspect A1 {
  after() : execution(int MathUtil.twice(int)) {
    System.out.println("MathUtil.twice executed");
  }
}
```

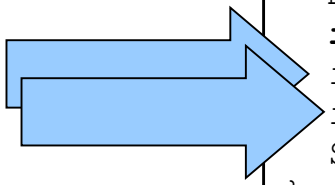
```
class Test {
  public static void main(String[] args) {
    MathUtil u = new MathUtil();
    int i = 2;
    i = u.twice(i);
    System.out.println(i);
  }
}
class MathUtil {
  public int twice(int i) {
    return i * 2;
  }
}
```



Syntax:
execution(ReturnType ClassName.Methodname(ParameterTypes))

EXECUTION VS CALL

```
aspect A1 {
  after() : call(int MathUtil.twice(int)) {
    System.out.println("MathUtil.twice called");
  }
}
```

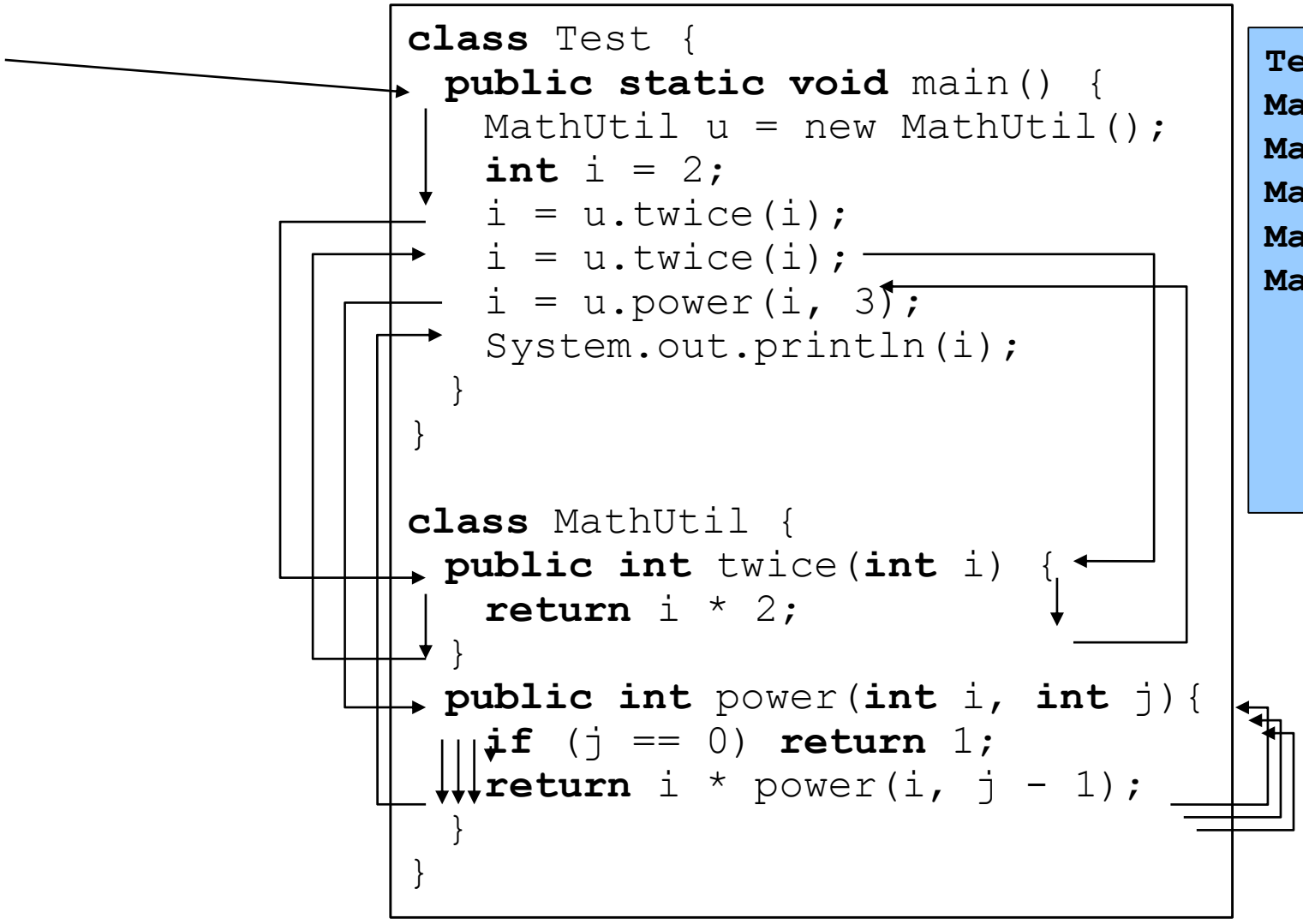


```
class Test {
public static void main(String[] args) {
  MathUtil u = new MathUtil();
  int i = 2;
  i = u.twice(i);
  i = u.twice(i);
  System.out.println(i);
}
}
class MathUtil {
public int twice(int i) {
  return i * 2;
}
}
```

PATTERN

```
aspect Execution {
  pointcut P1() : execution(int MathUtil.twice(int));

  pointcut P2() : execution(* MathUtil.twice(int));

  pointcut P3() : execution(int MathUtil.twice(*));

  pointcut P4() : execution(int MathUtil.twice(..));

  pointcut P5() : execution(int MathUtil.*(int, ..));

  pointcut P6() : execution(int *Util.tw*(int));

  pointcut P7() : execution(int *.twice(int));

  pointcut P8() : execution(int MathUtil+.twice(int));

  pointcut P9() : execution(public int package.MathUtil.twice(int)
    throws ValueNotSupportedException);

  pointcut Ptypisch() : execution(* MathUtil.twice(..));
}
```

CONTROL FLOW



Stack:

```
Test.main
MathUtil.twice
MathUtil.power
MathUtil.power
MathUtil.power
MathUtil.power
```

CFLOW

```
before() :
execution(* *.*(..))
```

```
execution(void Test.main(String[]))
execution(int MathUtil.twice(int))
execution(int MathUtil.twice(int))
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
```

```
execution(* *.*(..)) &&
 cflow(execution(* *.power(..)))
```

```
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
```

```
execution(* *.*(..)) &&
 cflowbelow(execution(* *.power(..)))
```

```
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
execution(int MathUtil.power(int, int))
```

```
execution(* *.power(..)) &&
 !cflowbelow(execution(* *.power(..)))
```

```
execution(int MathUtil.power(int, int))
```

GRAPH EXAMPLE

Basic Graph

```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

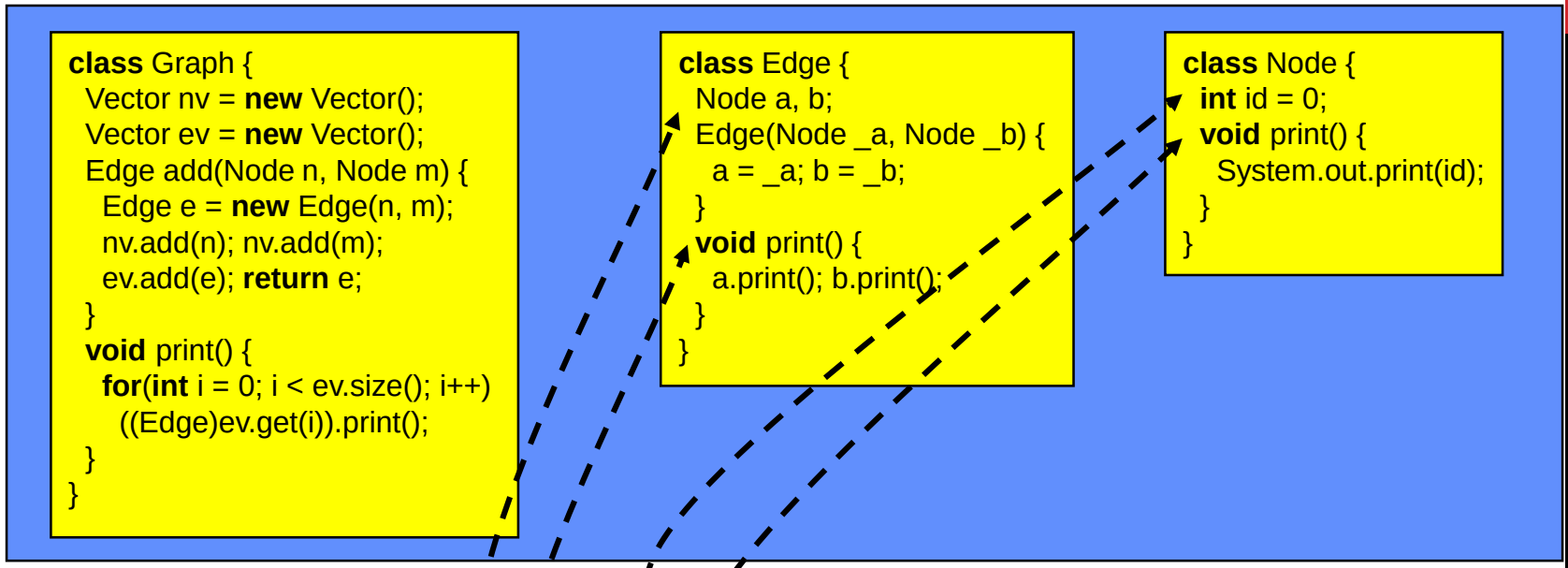
```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```

Color

```
aspect ColorAspect {
  Color Node.color = new Color();
  Color Edge.color = new Color();
  before(Node c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  before(Edge c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  static class Color { ... }
}
```

GRAPH EXAMPLE

Basic Graph



```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```

Color

```
aspect ColorAspect {
  static class Colored { Color color; }
  declare parents: (Node || Edge) extends Colored;
  before(Colored c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  static class Color { ... }
}
```

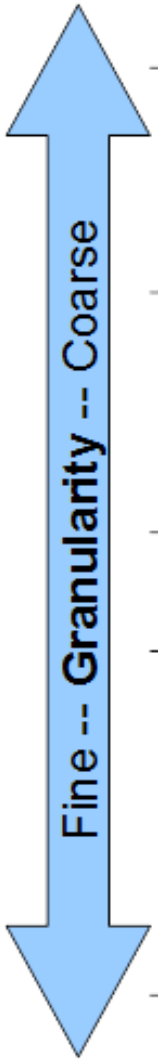
```
aspect Profiler {
    /** record time to execute my public methods */
    Object around() : execution(public * com.company..*.* (..)) {
        long start = System.currentTimeMillis();
        try {
            return proceed();
        } finally {
            long end = System.currentTimeMillis();
            printDuration(start, end,
                thisJoinPoint.getSignature());
        }
    }
    // implement recordTime...
}
```

```
aspect ConnectionPooling {
    ...
    Connection around() : call(Connection.new()) {
        if (enablePooling)
            if (!connectionPool.isEmpty())
                return connectionPool.remove(0);
        return proceed();
    }
    void around(Connection conn) :
         call(void Connection.close()) && target(conn) {
        if (enablePooling) {
            connectionPool.put(conn);
        } else {
            proceed();
        }
    }
}
```

```
abstract class Shape {
    abstract void moveBy(int x, int y);
}
class Point extends Shape { ... }
class Line extends Shape {
    Point start, end;
    void moveBy(int x, int y) { start.moveBy(x,y); end.moveBy(x,y); }
}

aspect DisplayUpdate {
    pointcut shapeChanged() : execution(void Shape+.moveBy(..));

    after() : shapeChanged() && !cflowbelow(shapeChanged()) {
        Display.update();
    }
}
```

```
aspect Autosave {
    int count = 0;
    after(): call(* Command+.execute(..)) {
        count++;
    }
    after(): call(* Application.save())
         || call(* Application.autosave()) {
        count = 0;
    }
    before(): call (* Command+.execute(..)) {
        if (count > 4) Application.autosave();
    }
}
```

Granularity of Extensions



- Add files

Simple SPL Tools

- Add modules/components/classes
- Explicit extension points

Components, Frameworks

- Add methods/fields
- Extend methods

Aspects, AHEAD, MDSoC

- Insert statement into middle of method

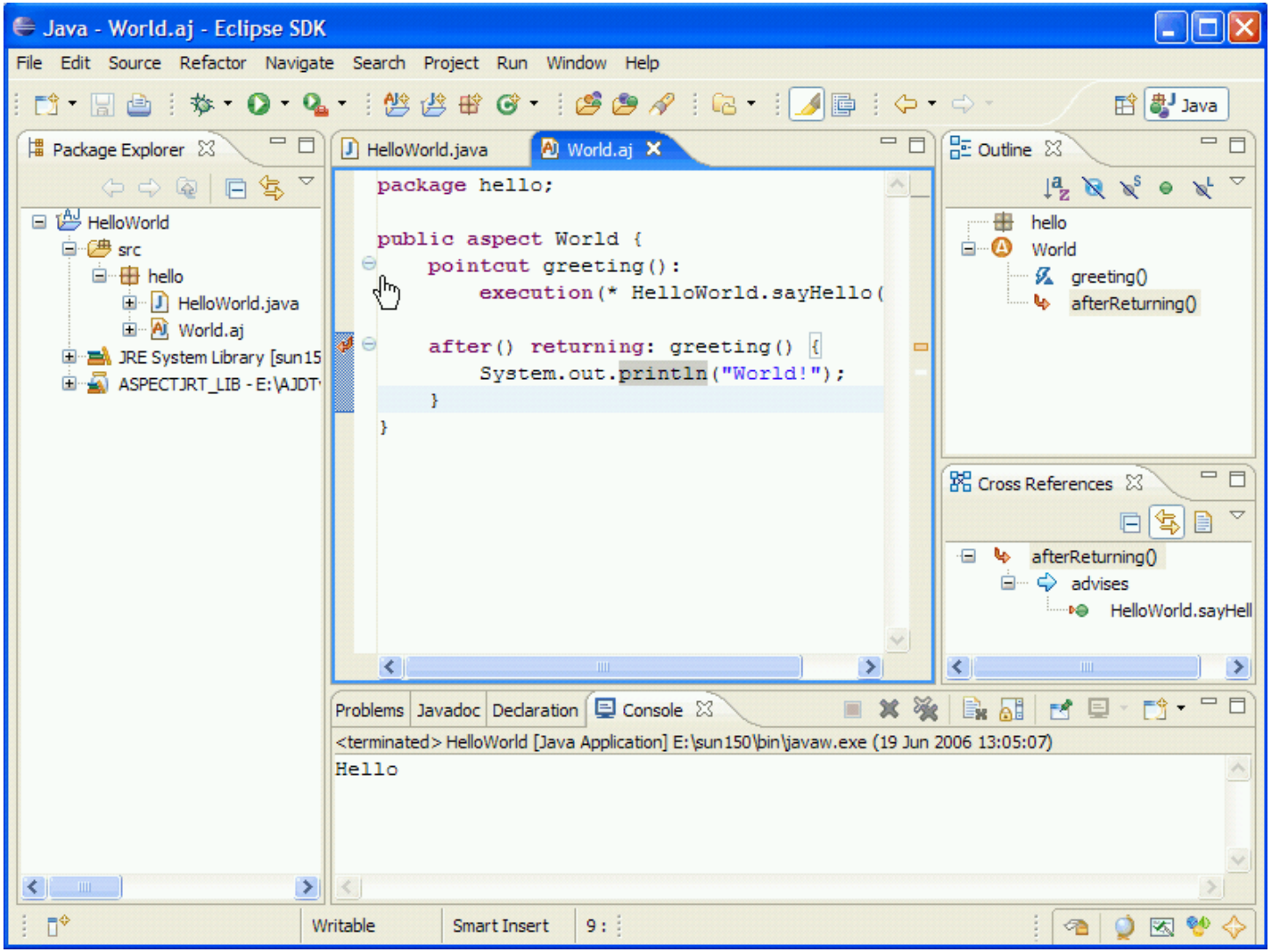
Aspects

- Extend expression
- Change method signature
- Insert Tokens

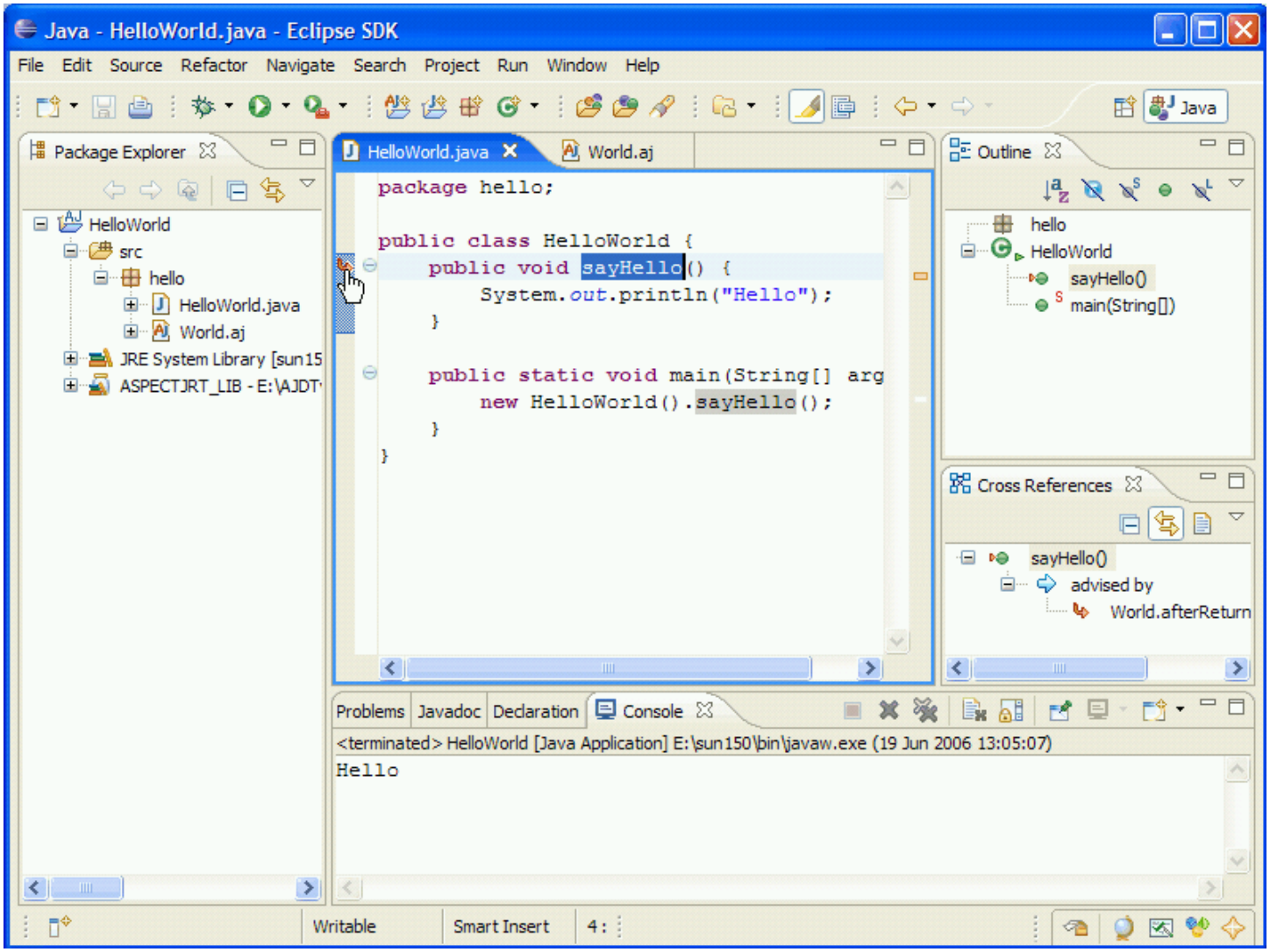
Preprocessors, Frames

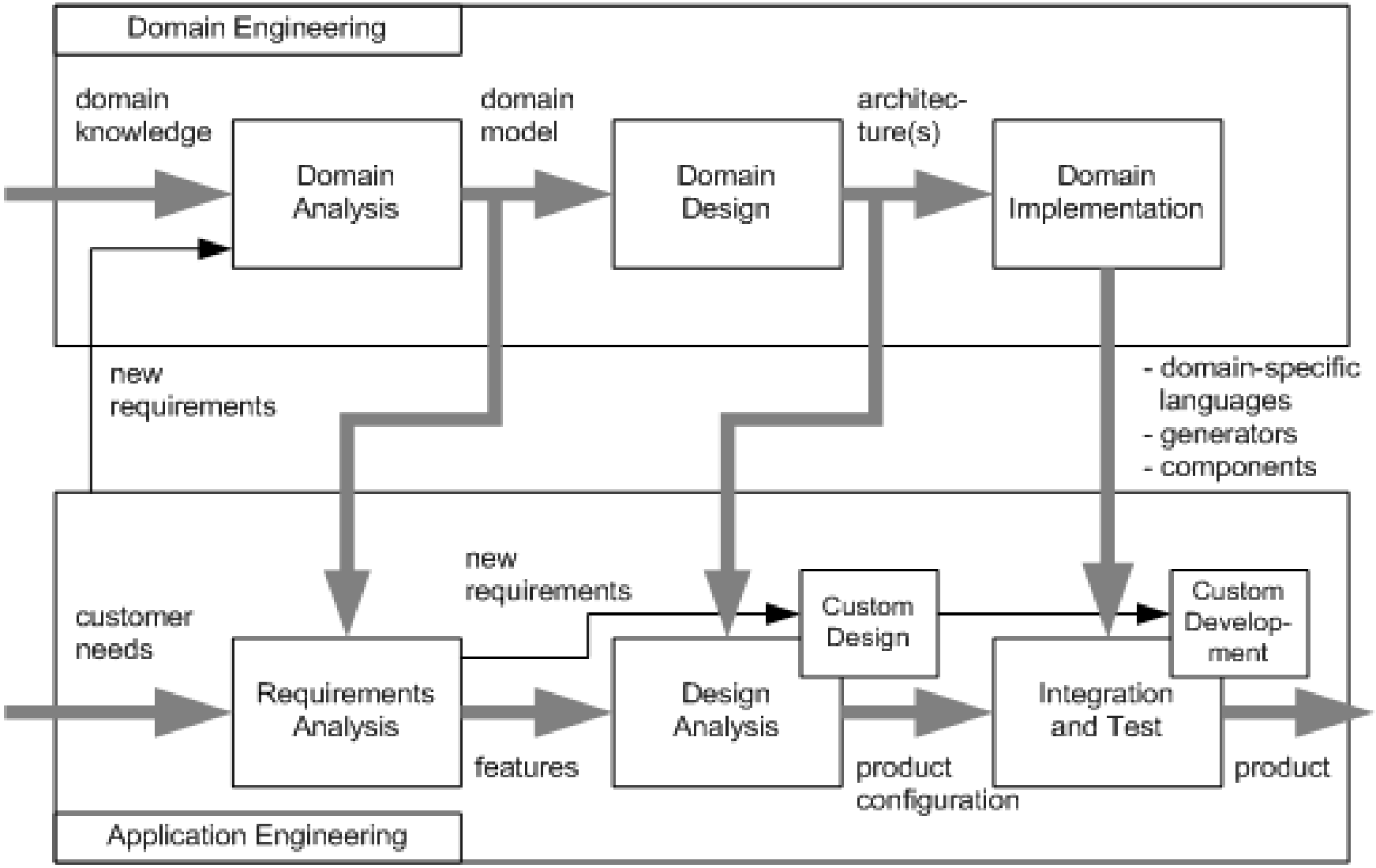
- Insert Characters

AJDT


Java - World.aj - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
HelloWorld
src
hello
HelloWorld.java
World.aj
JRE System Library [sun15
ASPECTJRT_LIB - E:\AJDT
HelloWorld.java
World.aj
package hello;
public aspect World {
pointcut greeting():
execution(* HelloWorld.sayHello(
after() returning: greeting() {
System.out.println("World!");
}
}
Outline
hello
World
greeting()
afterReturning()
Cross References
afterReturning()
advises
HelloWorld.sayHell
Problems Javadoc Declaration Console
<terminated> HelloWorld [Java Application] E:\sun150\bin\javaw.exe (19 Jun 2006 13:05:07)
Hello
Writable
Smart Insert
9 :

AJDT


Java - HelloWorld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Package Explorer
HelloWorld
src
hello
HelloWorld.java
World.aj
JRE System Library [sun15
ASPECTJRT_LIB - E:\AJDT
HelloWorld.java
World.aj
package hello;
public class HelloWorld {
public void sayHello() {
System.out.println("Hello");
}
public static void main(String[] arg
new HelloWorld().sayHello();
}
}
Outline
hello
HelloWorld
sayHello()
main(String[])
Cross References
sayHello()
advised by
World.afterReturn
Problems Javadoc Declaration Console
<terminated> HelloWorld [Java Application] E:\sun150\bin\javaw.exe (19 Jun 2006 13:05:07)
Hello
Writable
Smart Insert
4 :


Domain Engineering
domain knowledge
Domain Analysis
domain model
Domain Design
architec-ture(s)
Domain Implementation
new requirements
- domain-specific languages
- generators
- components
customer needs
Requirements Analysis
new requirements
Custom Design
Design Analysis
Custom Develop-ment
Integration and Test
features
product configuration
product
Application Engineering

DISCUSSION

DISCUSSION

Obliviousness

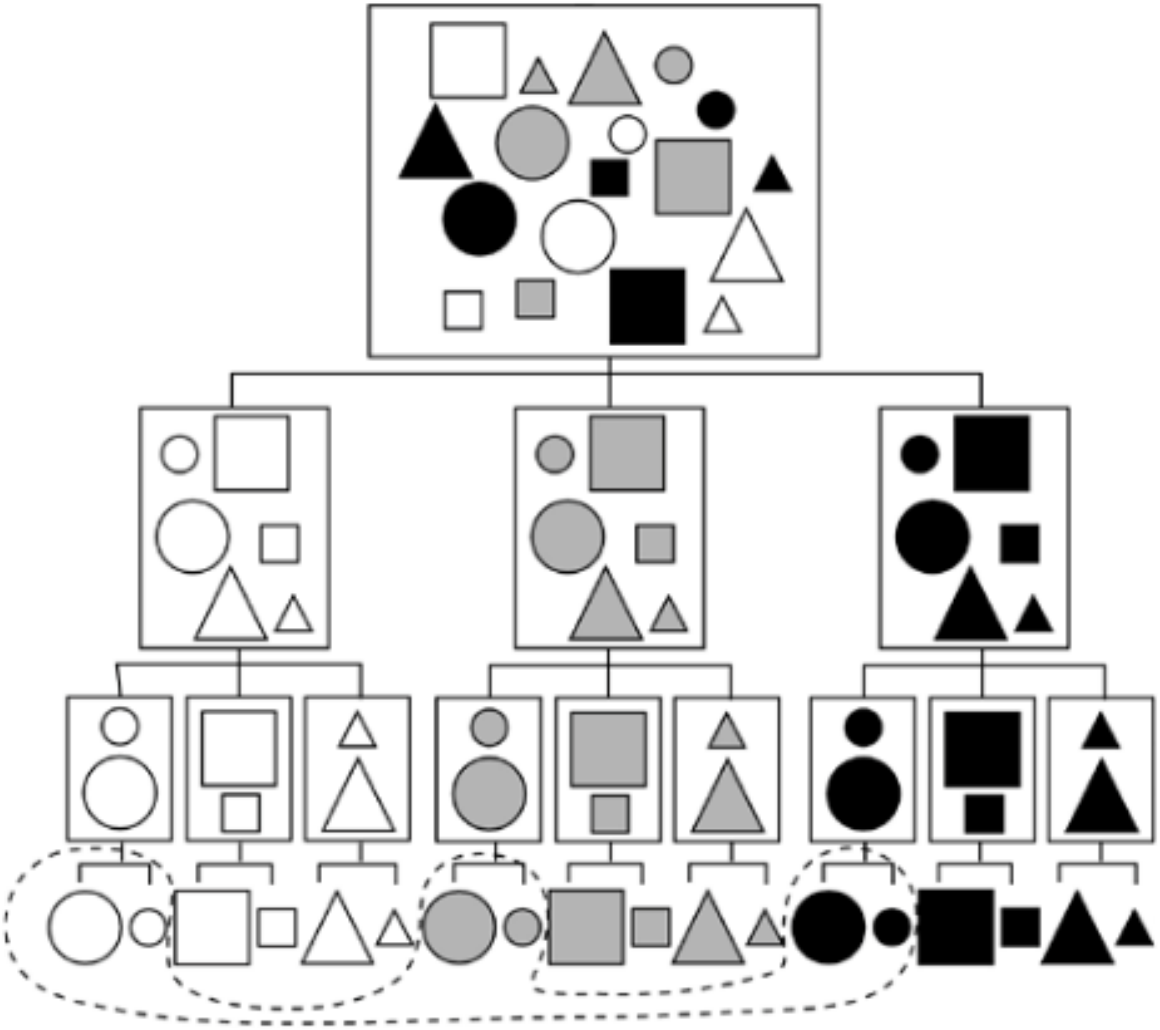
Quantification

Separation of Concerns

Information Hiding

Uniformity

Tyranny of the dominant decomposition

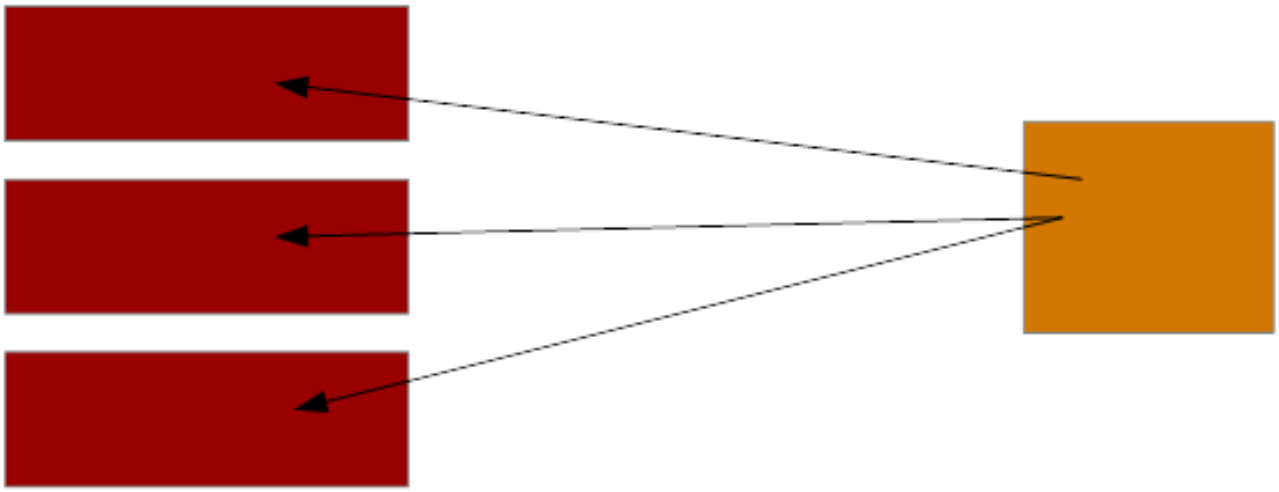


[Tarr et al., 19

FRAGILE POINTCUTS

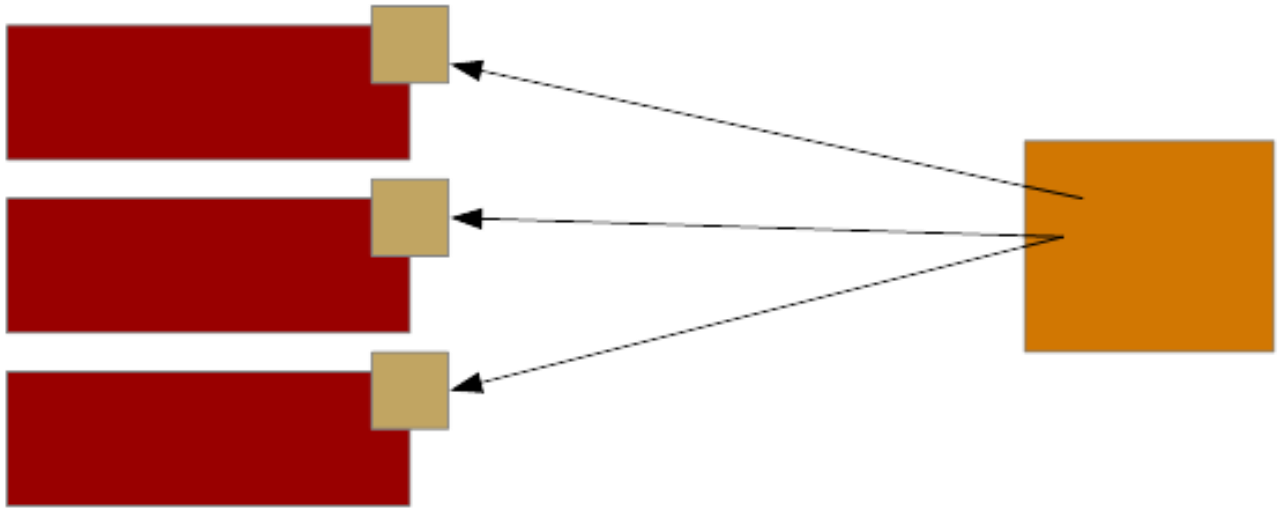
```
class Chess {
    void drawKing() {...}
    void drawQueen() {...}
    void drawKnight() {...}
}

aspect UpdateDisplay {
    pointcut drawn : execution(* draw*(..));
    ...
}
```

AspectJ



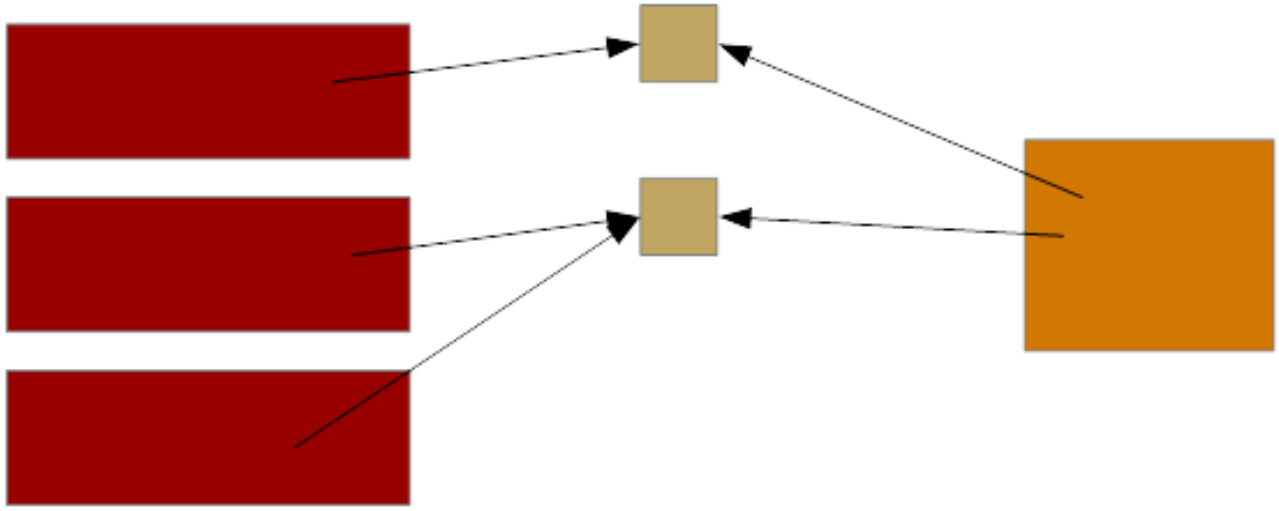
privileged aspects may even affect private join points

Open Modules



[Aldrich, 2005]

IIIA, ...



[Steimann et al. 2010]

COMPLICATED SYNTAX

OOP / FOP

```
public void delete(Transaction txn, DbEntry key) {
    super.delete(txn, key);
    Tracer.trace(Level.FINE, "Db.delete", this, txn, key);
}
```

AOP

```
pointcut traceDel(Database db, Transaction txn, DbEntry key) :
    execution(void Database.delete(Transaction, DbEntry))
    && args(txn, key) && within(Database) && this(db);
after(Database db, Transaction txn, DbEntry key): traceDel(db, txn, key) {
    Tracer.trace(Level.FINE, "Db.delete", db, txn, key);
}
```

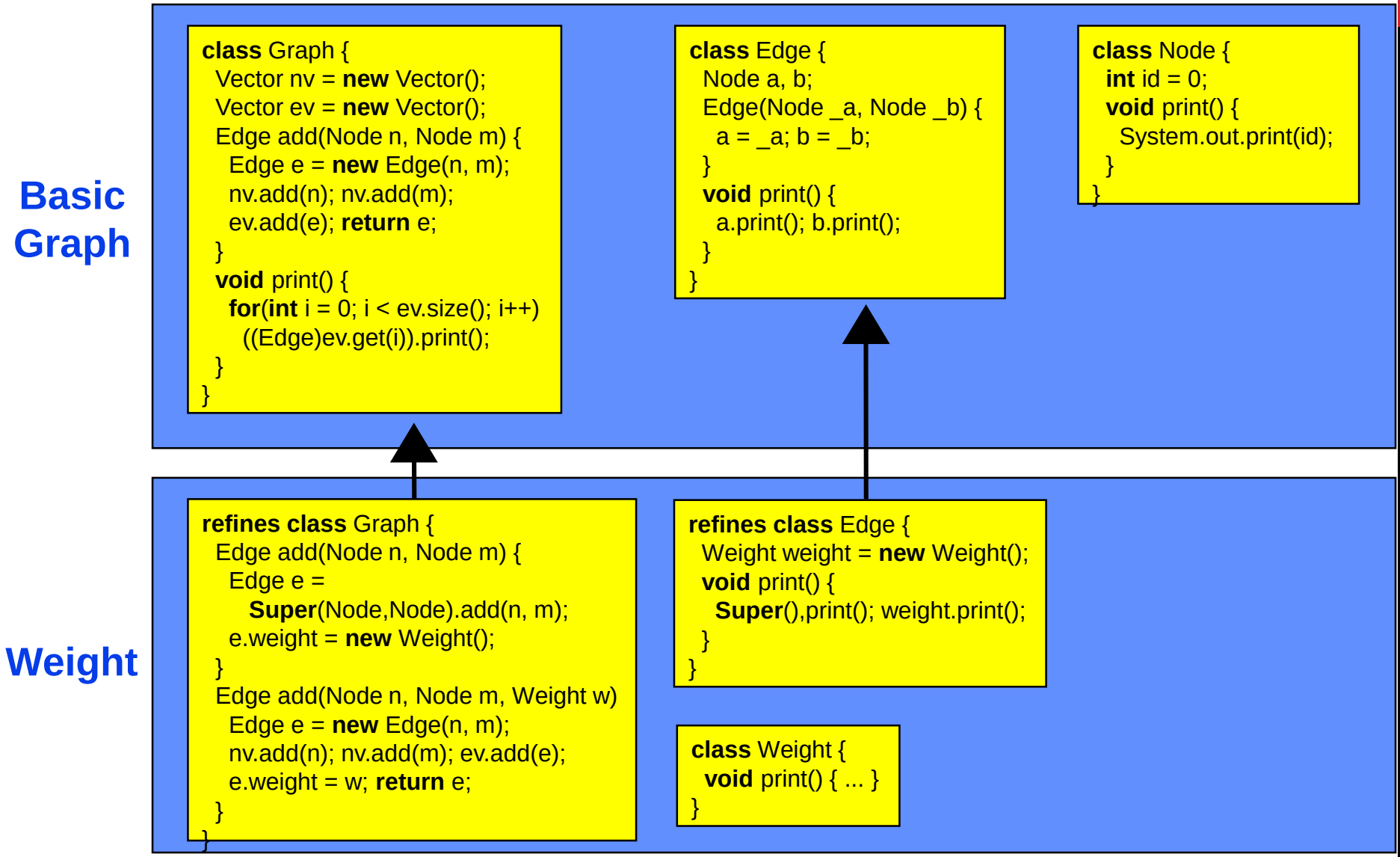
AOP VS FOP

Basic Graph

```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```


Basic Graph
class Graph {
Vector nv = new Vector();
Vector ev = new Vector();
Edge add(Node n, Node m) {
Edge e = new Edge(n, m);
nv.add(n); nv.add(m);
ev.add(e); return e;
}
void print() {
for(int i = 0; i < ev.size(); i++)
((Edge)ev.get(i)).print();
}
}
class Edge {
Node a, b;
Edge(Node _a, Node _b) {
a = _a; b = _b;
}
void print() {
a.print(); b.print();
}
}
class Node {
int id = 0;
void print() {
System.out.print(id);
}
}
Weight
refines class Graph {
Edge add(Node n, Node m) {
Edge e =
Super(Node,Node).add(n, m);
e.weight = new Weight();
}
Edge add(Node n, Node m, Weight w)
Edge e = new Edge(n, m);
nv.add(n); nv.add(m); ev.add(e);
e.weight = w; return e;
}
}
refines class Edge {
Weight weight = new Weight();
void print() {
Super().print(); weight.print();
}
}
class Weight {
void print() { ... }
}

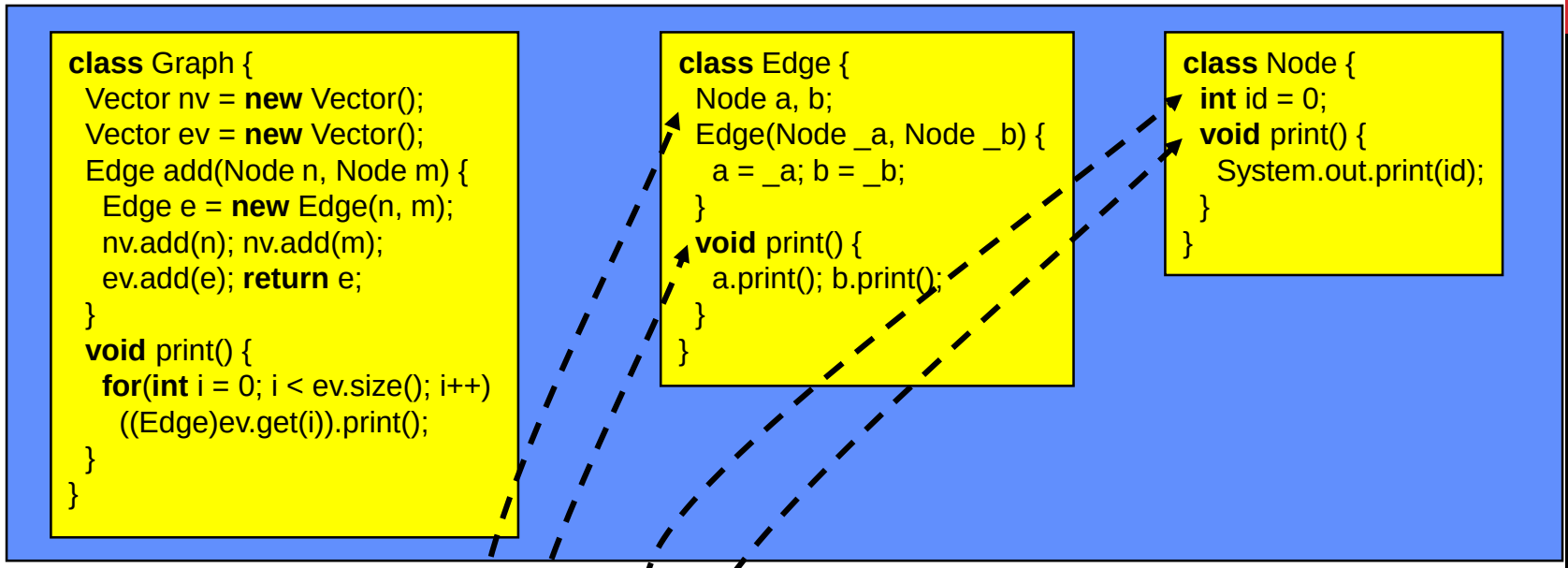
Basic Graph

```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```

Basic Graph



```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

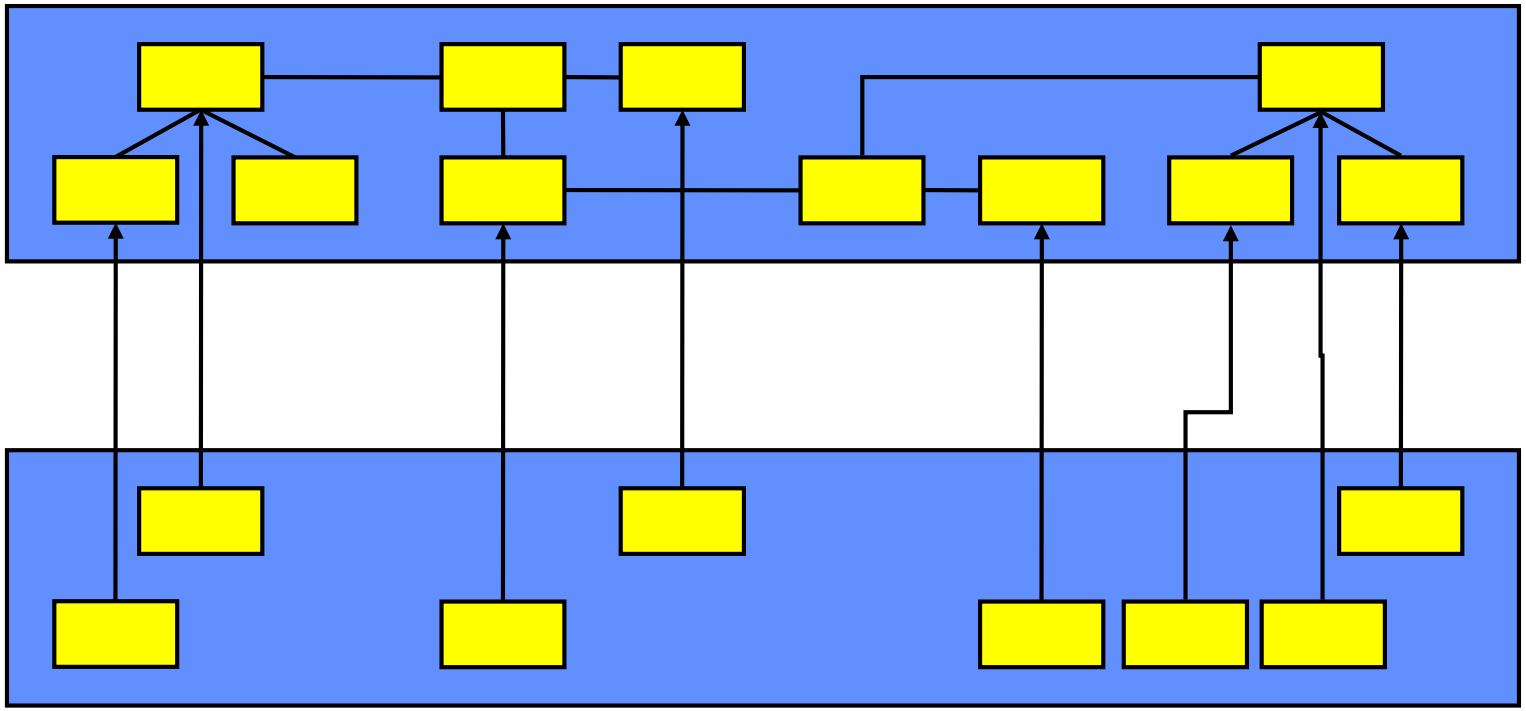
```
class Edge {
  Node a, b;
  Edge(Node _a, Node _b) {
    a = _a; b = _b;
  }
  void print() {
    a.print(); b.print();
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    System.out.print(id);
  }
}
```

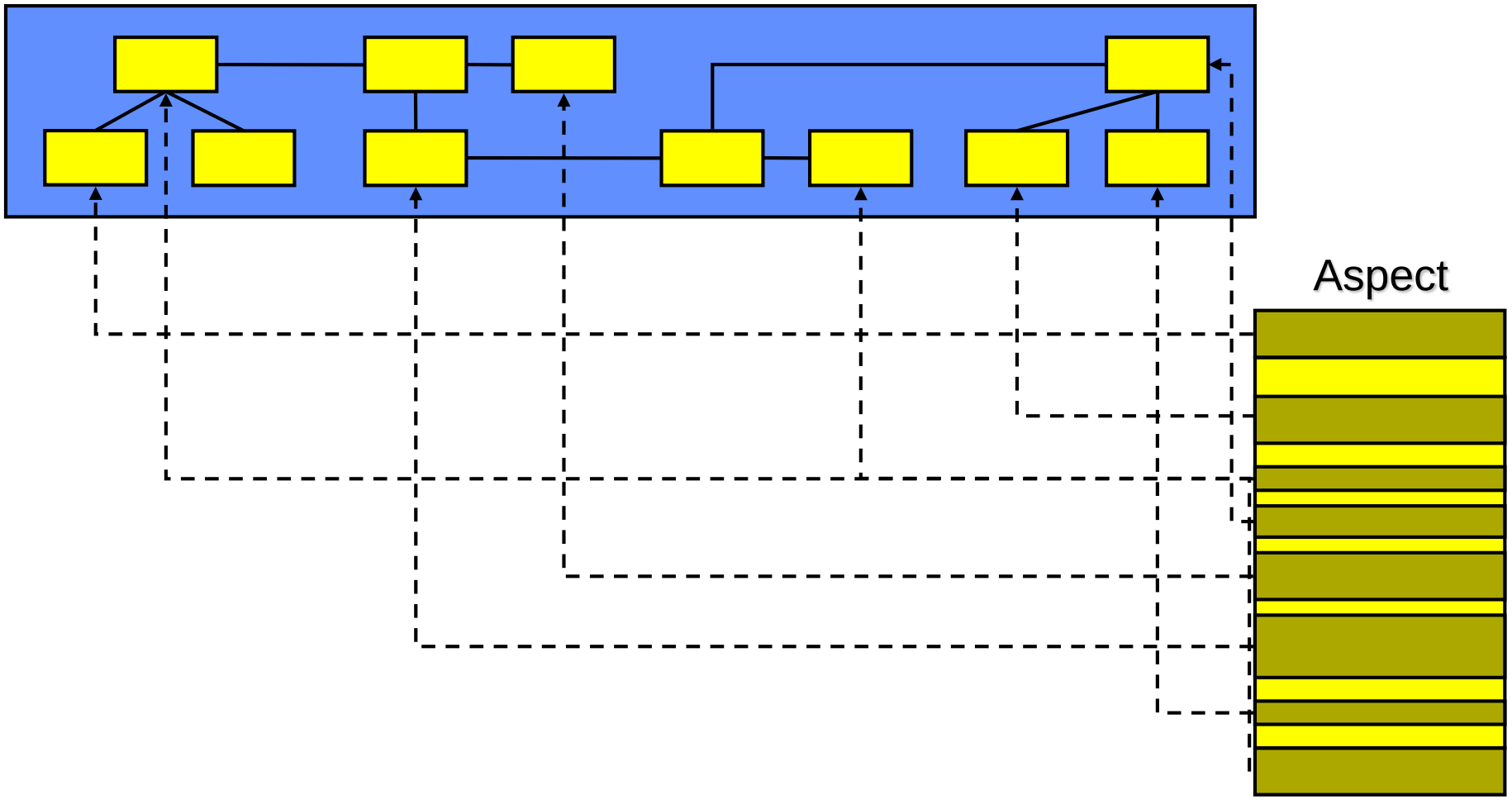
Color

```
aspect ColorAspect {
  static class Colored { Color color; }
  declare parents: (Node || Edge) extends Colored;
  before(Colored c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  static class Color { ... }
}
```

AOP VS. FOP



AOP VS. FOP



HOMOGENEOUS VS HETEROGENEOUS

```
class Graph {  …
    Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = new Weight(); return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m); ev.add(e);
    e.weight = w; return e;
  }  …
}
```

```
class Edge {  …
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
   void print() {
    a.print(); b.print(); weight.print();
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
  }
}
```

STATIC VS DYNAMIC

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    System.out.print(id);
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

SIMPLE DYNAMIC EXTENSIONS

```
class Edge {
  int weight = 0;
  void setWeight(int w) { weight = w; }
  int getWeight() { return weight; }
}
```

Jak

```
refines class Edge {
  void setWeight(int w) {
    Super(int).setWeight(2*w);
  }

  int getWeight() {
    return Super().getWeight()/2;
  }
}
```

AspectJ

```
aspect DoubleWeight {
  void around(int w) : args(w) &&
    execution(void Edge.setWeight(int)) {
      proceed(w*2);
  }
  int around() :
    execution(void Edge.getWeight()) {
      return proceed()/2;
  }
}
```

COMPLEX DYNAMIC EXTENSIONS

```
class Node {
  void print() {...}
  ...
}
```

Jak

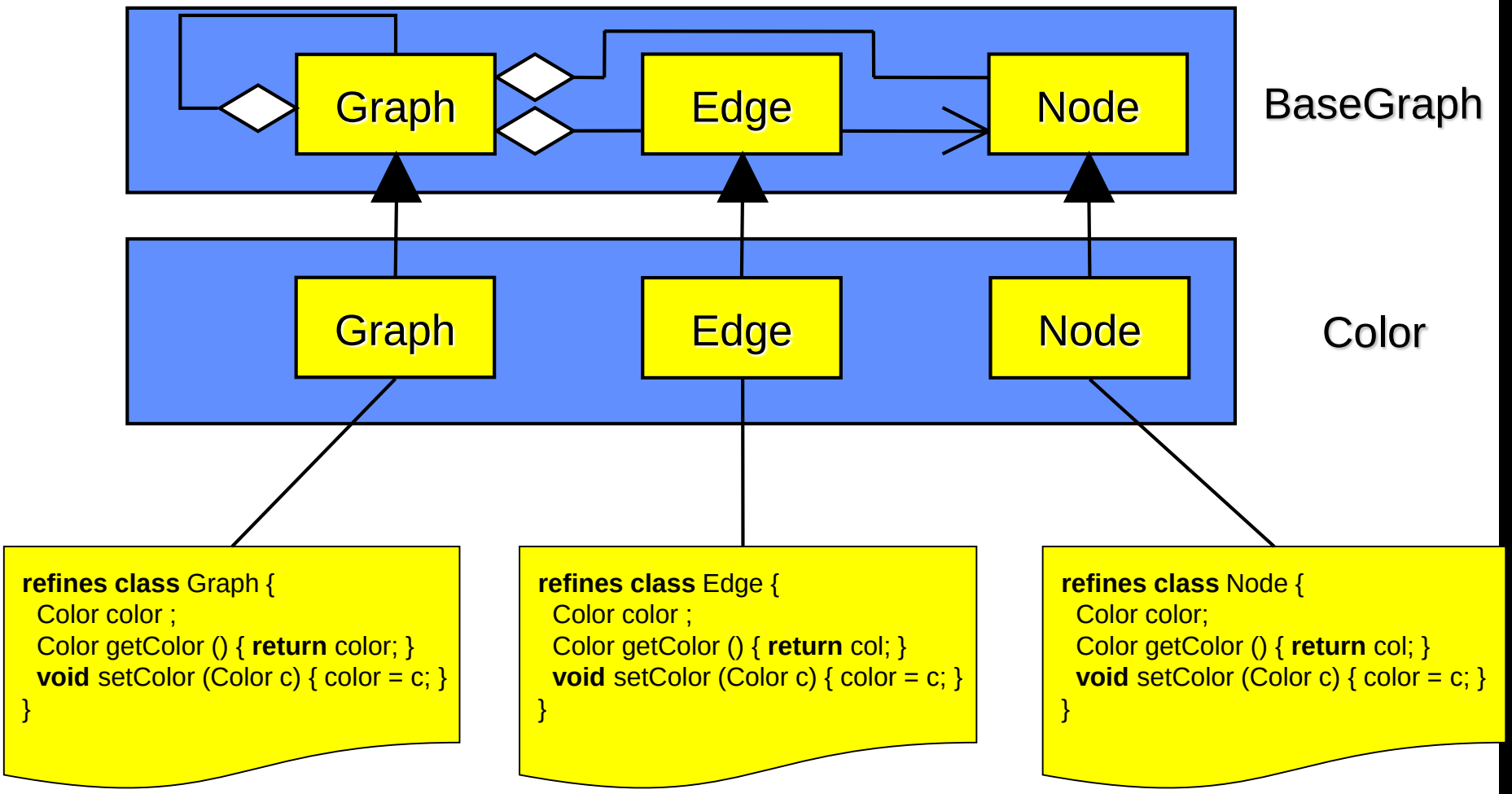
```
refines class Node {
  static int count = 0;
  void print() {
    if(count == 0)
      printHeader();
    count++;
    Super().print();
    count--;
  }
  void printHeader() { /* ... */ }
}
```

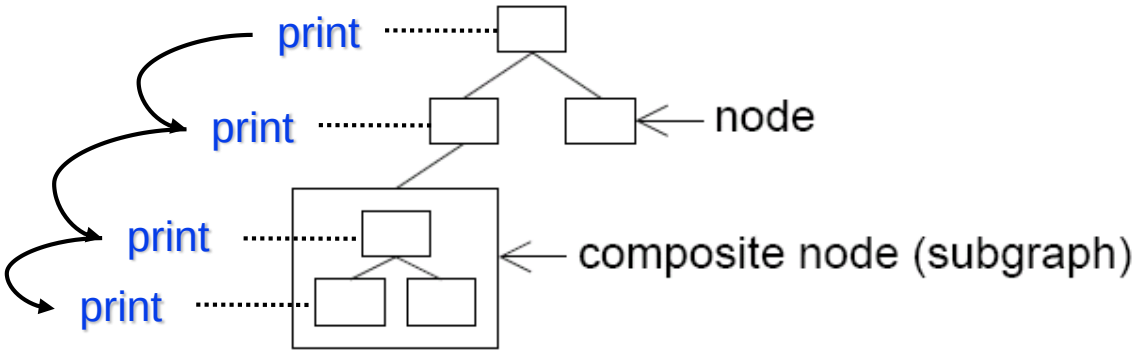
AspectJ

```
aspect PrintHeader {
  before() :
    execution(void print()) &&
    !cflowbelow(execution(void print())) {
      printHeader();
  }
  void printHeader() { /* … */ }
}
```

COMPARISON

	FOP	AOP
static		
dynamic		
heterog.		
homog.		


Graph
Edge
Node
BaseGraph
Graph
Edge
Node
Color
refines class Graph {
Color color ;
Color getColor () { return color; }
void setColor (Color c) { color = c; }
}
refines class Edge {
Color color ;
Color getColor () { return col; }
void setColor (Color c) { color = c; }
}
refines class Node {
Color color;
Color getColor () { return col; }
void setColor (Color c) { color = c; }
}

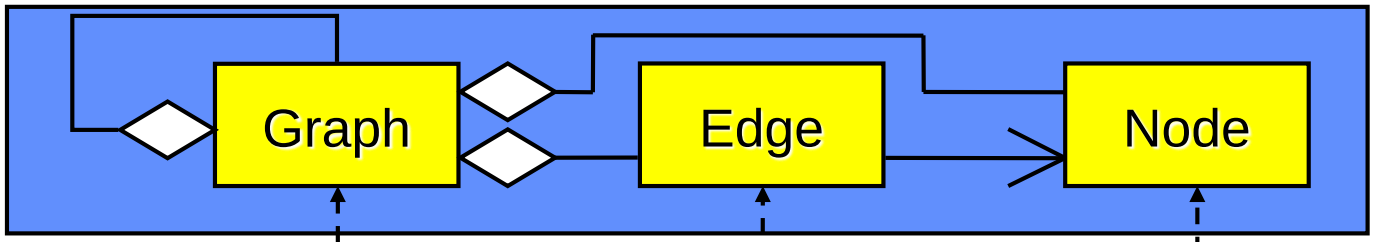


AspectJ

```
aspect PrintHeader {
  before() : execution(void print ()) &&
    !cflowbelow (execution(void print ())) {
      printHeader ();
  }
  void printHeader () { /* … */ }
}
```

Jak

```
refines class Node {
  static int count = 0;
  void print () {
    if(count == 0) printHeader ();
    count++; Super().print (); count--;
  }
  void printHeader () { /* ... */ }
}
```

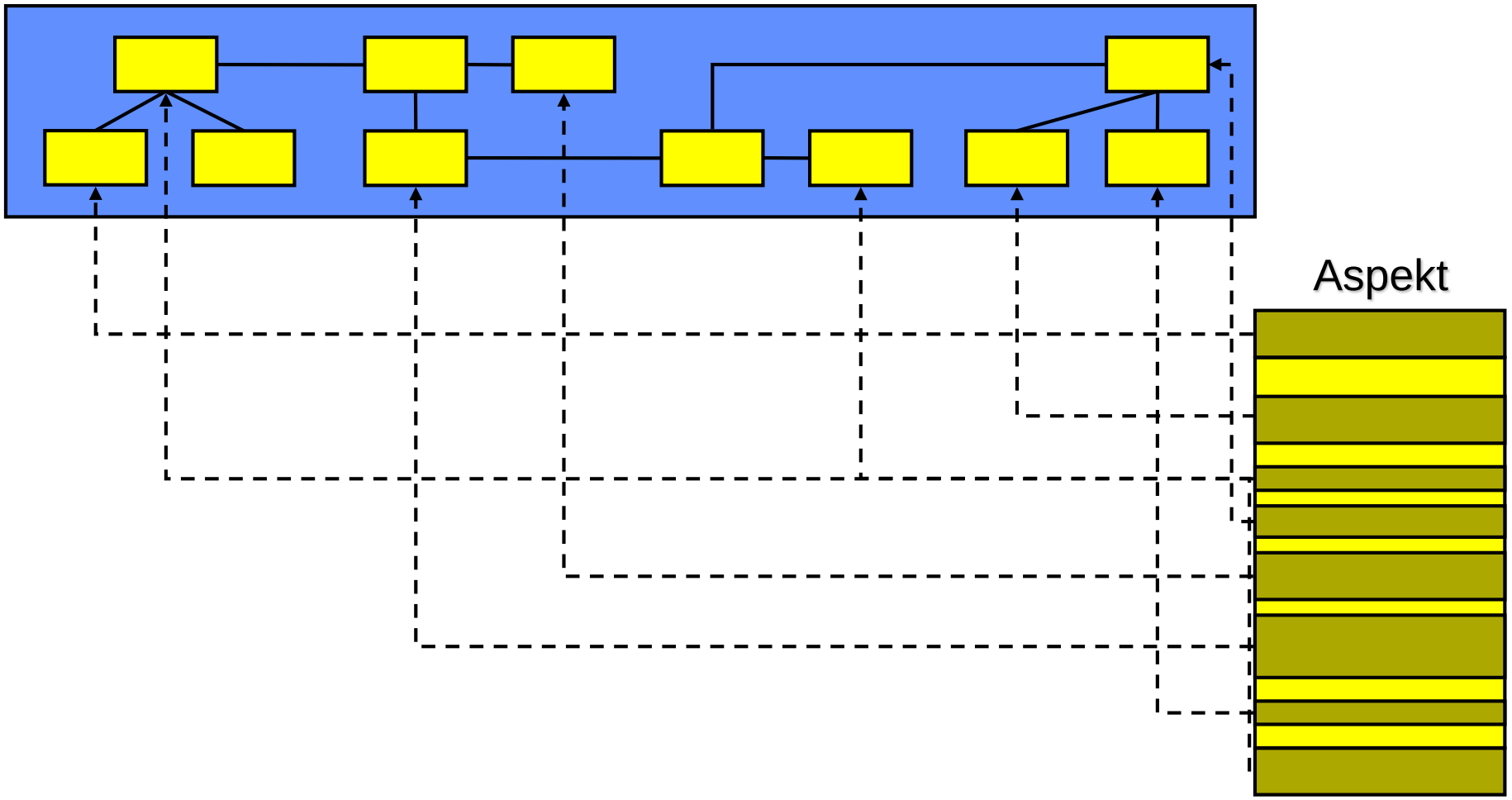


BaseGraph

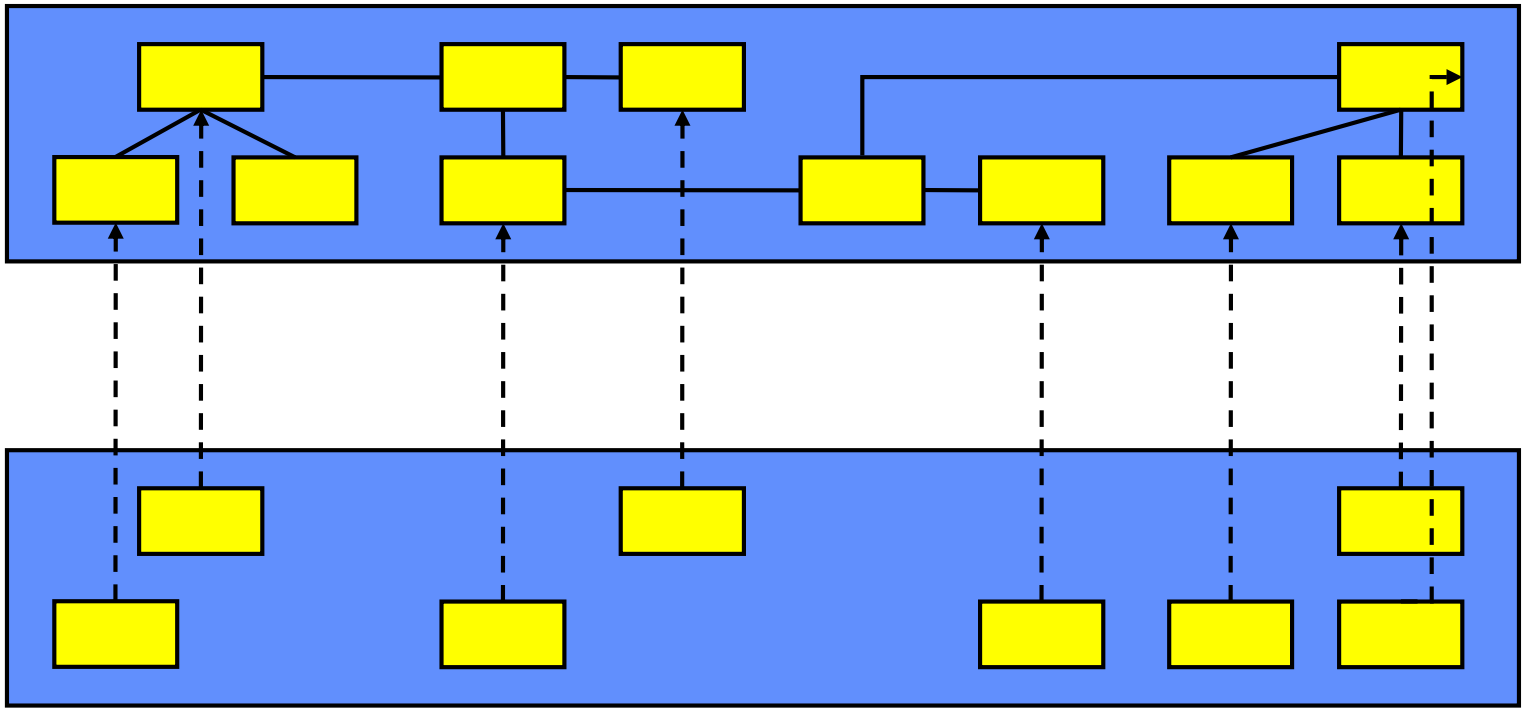
```
aspect AddWeight {
  Edge Graph.add(Node n, Node m, Weight w) {
    Edge res = add(n, m); res.weight = w; return res;
  }
  Weight Edge.weight ;
  after(Edge e) : this(e) &&
    execution(void Edge.print ()) {  /* … */  }
}
```

object structure no longer explicit

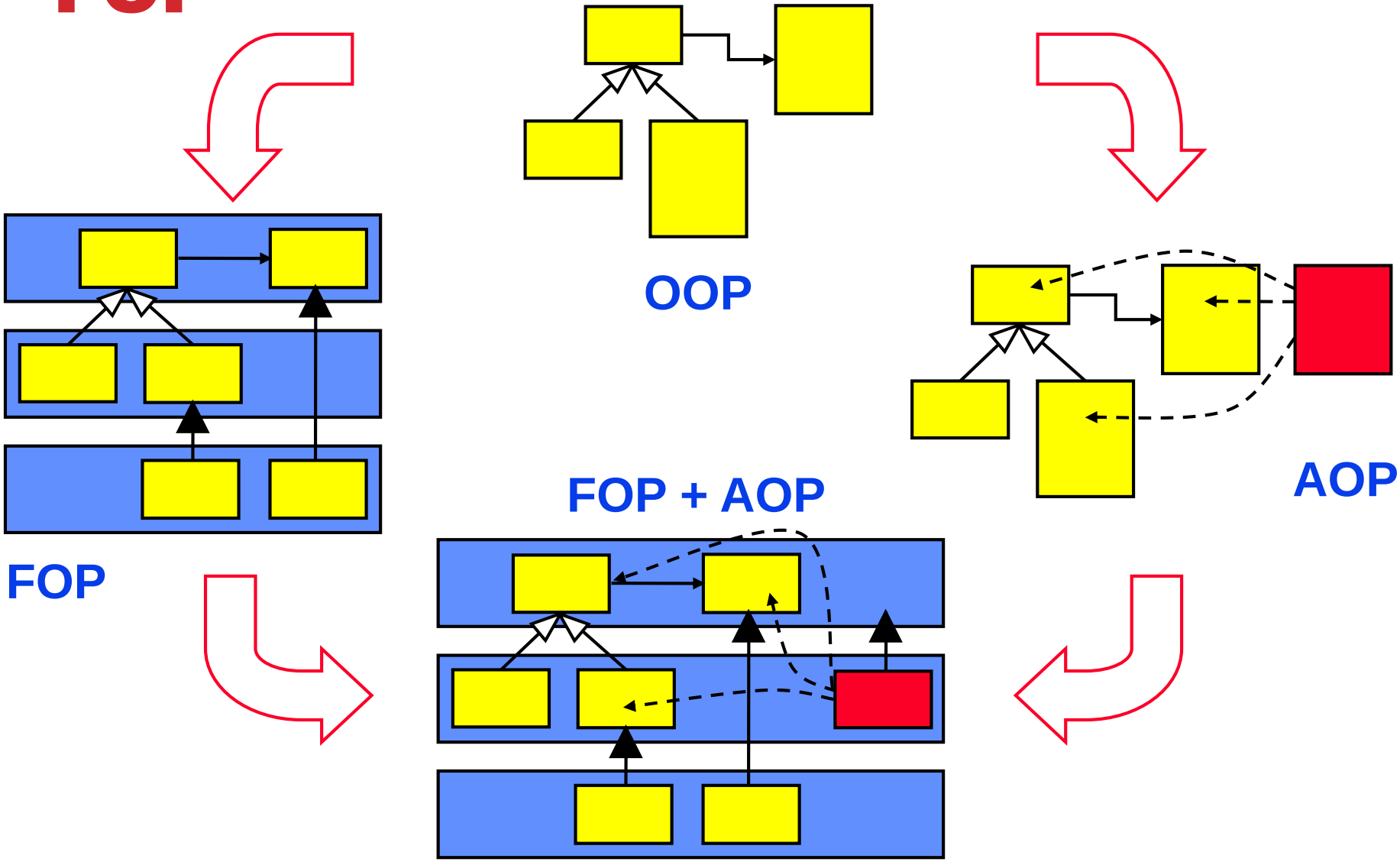
SCALABILITY



ONE ASPECT PER ROLE



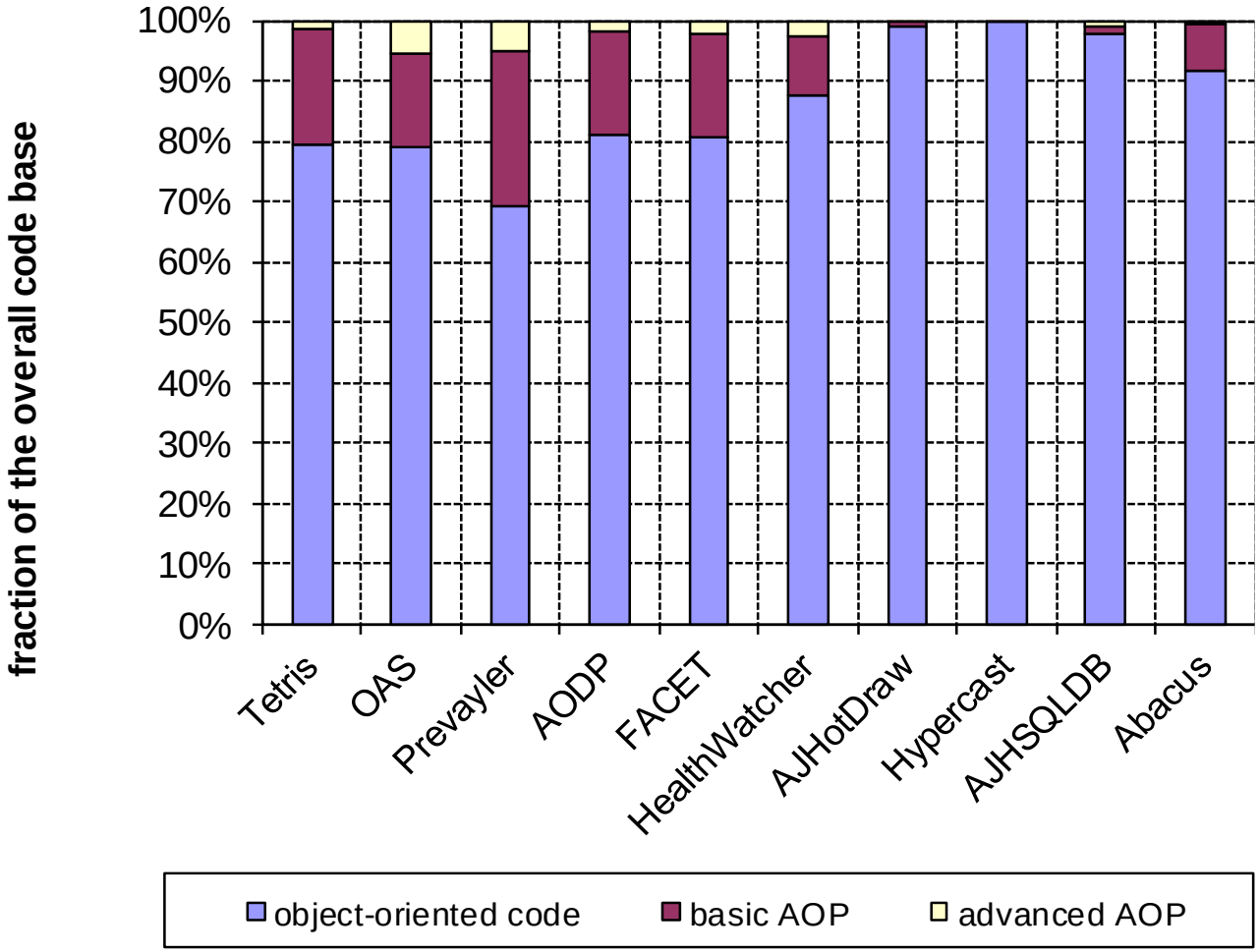
COMBINING AOP AND FOP



ASPECTJ PROGRAMS

Tetris	The popular game	1 KLOC	Blekinge Institute of Technology
OAS	An online auction system	2 KLOC	Lancaster University
Prevayler	Transparent persistence for Java	4 KLOC	University of Toronto
AODP	Aspect-oriented implementation of the Gang-of-Four design patterns	4 KLOC	University of British Columbia
FACET	An aspect-based CORBA event channel	6 KLOC	Washington University
HealthWatcher	Web-based information system for public health systems	7 KLOC	Lancaster University
AJHotDraw	2D graphics framework	22 KLOC	Sourceforge project
Hypercast	Multicast overlay network communication	67 KLOC	University of Virginia, Microsoft
AJHSQLDB	SQL relational database engine	76 KLOC	University of Passau
Abacus	A CORBA middleware framework	130 KLOC	University of Toronto

AOP USE


fraction of the overall code base
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Tetris
OAS
Prevayler
AODP
FACET
HealthWatcher
AJHotDraw
Hypercast
AJHSQLDB
Abacus
object-oriented code
basic AOP
advanced AOP

FURTHER READING

eclipse.org/aspectj, eclipse.org/ajdt

R. Laddad. AspectJ in Action: Practical Aspect-Oriented Programming. Manning Publications, 2003.
R.E. Filman and D.P. Friedman, Aspect-Oriented Programming is Quantification and Obliviousness, In Workshop on Advanced Separation of Concerns, OOPSLA 2000
S. Apel. How AspectJ is Used: An Analysis of Eleven AspectJ Programs. Journal of Object Technology, 9(1), 2010.