The Eclipse Runtime Perspective for Object-Oriented Code Exploration and Program Comprehension

Marwan Abi-Antoun
Andrew Giang
Sumukhi Chandrashekar
Ebrahim Khalaj

Wayne State University
Department of Computer Science
Detroit, Michigan, USA

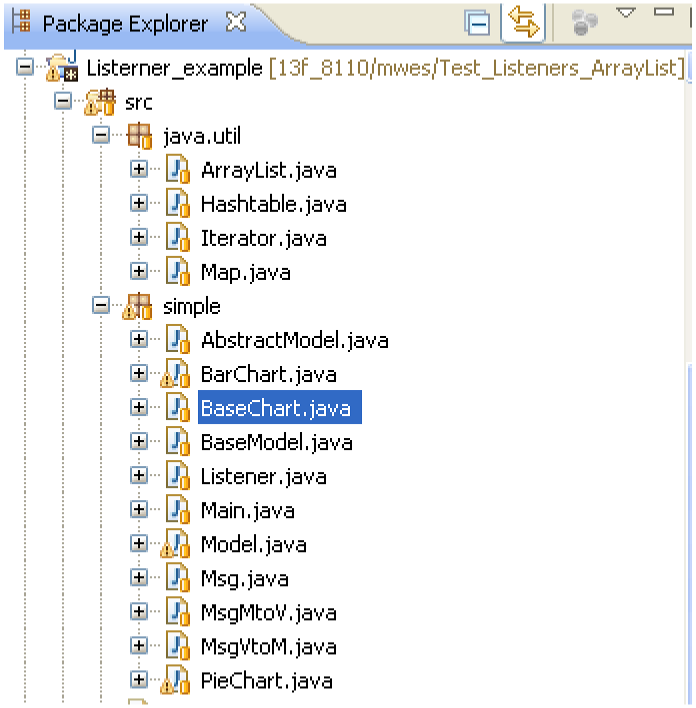
Motivation: A new Perspective

- IDEs emphasize design-time perspective based on code structure
  - Class-oriented view
  - Hierarchy of classes

Such views do not answer

- What are the architectural tiers of the application?
- Where is an instance of type A created?
- Can one access an instance of type A from an instance of type B?

Package Explorer

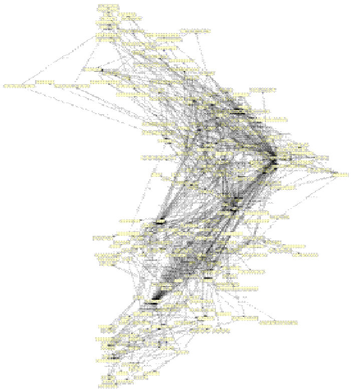


```
Package Explorer
Listerner_example [13f_8110/mwes/Test_Listeners_ArrayList]
  src
    java.util
      ArrayList.java
      Hashtable.java
      Iterator.java
      Map.java
    simple
      AbstractModel.java
      BarChart.java
      BaseChart.java
      BaseModel.java
      Listener.java
      Main.java
      Model.java
      Msg.java
      MsgMtoV.java
      MsgVtoM.java
      PieChart.java
```

Motivation: A new Perspective

- Another perspective: the runtime perspective
  - Eclipse debugger
    - Specific instances of type A may not matter
  - Object graphs:
    - Too large (without abstraction)
    - May not convey design intent
    - Need to apply abstraction

Flat object graph: Womble
[Jackson and Waingold '01]

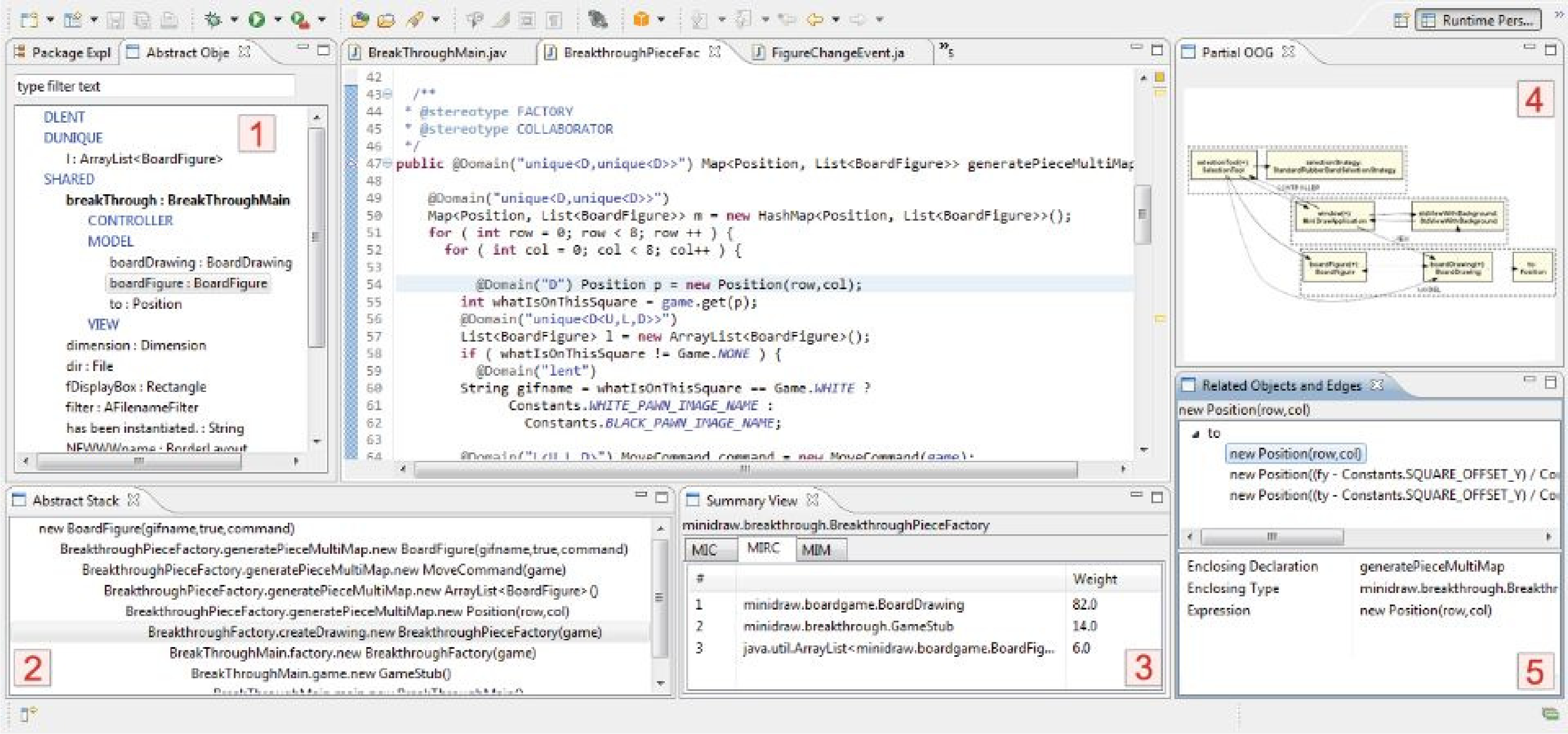


Motivation: A new Perspective

- Emphasize design-time perspective based on **abstract** runtime structure
  - Extracted using abstract interpretation
  - Hierarchy of abstract objects
    [Abi-Antoun and Aldrich, OOPSLA, 2009]
  - Summarization of runtime objects
  - Abstraction keeps graph manageable
- Use static analysis so tool works at design time

Motivation: A new Perspective

- New perspective integrated with Eclipse IDE



type filter text

DLENT
DUNIQUE
    l : ArrayList<BoardFigure>
SHARED
    breakThrough : BreakThroughMain
        CONTROLLER
        MODEL
            boardDrawing : BoardDrawing
            boardFigure : BoardFigure
            to : Position
        VIEW
    dimension : Dimension
    dir : File
    fDisplayBox : Rectangle
    filter : AFilenameFilter
    has been instantiated. : String
    NEWWWname : BorderLayout

```
 /**
* @stereotype FACTORY
* @stereotype COLLABORATOR
*/
public @Domain("unique<D,unique<D>>") Map<Position, List<BoardFigure>> generatePieceMultiMap

    @Domain("unique<D,unique<D>>")
    Map<Position, List<BoardFigure>> m = new HashMap<Position, List<BoardFigure>>();
    for ( int row = 0; row < 8; row ++ ) {
      for ( int col = 0; col < 8; col++ ) {

          @Domain("D") Position p = new Position(row,col);
        int whatIsOnThisSquare = game.get(p);
        @Domain("unique<D<U,L,D>>")
        List<BoardFigure> l = new ArrayList<BoardFigure>();
        if ( whatIsOnThisSquare != Game.NONE ) {
          @Domain("lent")
        String gifname = whatIsOnThisSquare == Game.WHITE ?
              Constants.WHITE_PAWN_IMAGE_NAME :
                Constants.BLACK_PAWN_IMAGE_NAME;

        @Domain("L<U,L,D>") MoveCommand command = new MoveCommand(game);
```

new Position(row,col)

- to
    - new Position(row,col)
    - new Position((fy - Constants.SQUARE_OFFSET_Y) / Co
    - new Position((ty - Constants.SQUARE_OFFSET_Y) / Co

Enclosing Declaration	generatePieceMultiMap
Enclosing Type	minidraw.breakthrough.Breakthr
Expression	new Position(row,col)

new BoardFigure(gifname,true,command)
BreakthroughPieceFactory.generatePieceMultiMap.new BoardFigure(gifname,true,command)
BreakthroughPieceFactory.generatePieceMultiMap.new MoveCommand(game)
BreakthroughPieceFactory.generatePieceMultiMap.new ArrayList<BoardFigure>()
BreakthroughPieceFactory.generatePieceMultiMap.new Position(row,col)
BreakthroughFactory.createDrawing.new BreakthroughPieceFactory(game)
BreakThroughMain.factory.new BreakthroughFactory(game)
BreakThroughMain.game.new GameStub()
BreakThroughMain.main.new BreakThroughMain()

minidraw.breakthrough.BreakthroughPieceFactory

MIC | MIRC | MIM

#		Weight
1	minidraw.boardgame.BoardDrawing	82.0
2	minidraw.breakthrough.GameStub	14.0
3	java.util.ArrayList<minidraw.boardgame.BoardFig...	6.0

Contributions

- Novel Eclipse development-time perspective
  - focuses on abstract runtime structure
  - complements existing perspectives
  - displays information to developers using diagrammatic and non-diagrammatic views

Outline

- Extract abstract runtime structure
- Task centric demonstration of ArchDoc
- Contrast Java perspective with ArchDoc
- Potential applications of ArchDoc

Extracting abstract runtime structure

- Add annotations and type check them
  - Express architectural hierarchy
  - Annotations must match the code
- Extract hierarchical abstract object graph using static analysis
  - Architecturally relevant objects at the top level
  - Abstract edges connect abstract objects
- Save abstract graph to external file
- Switch to Runtime Perspective (ArchDoc)
  - Mines the hierarchical abstract graph
  - Displays information in various view

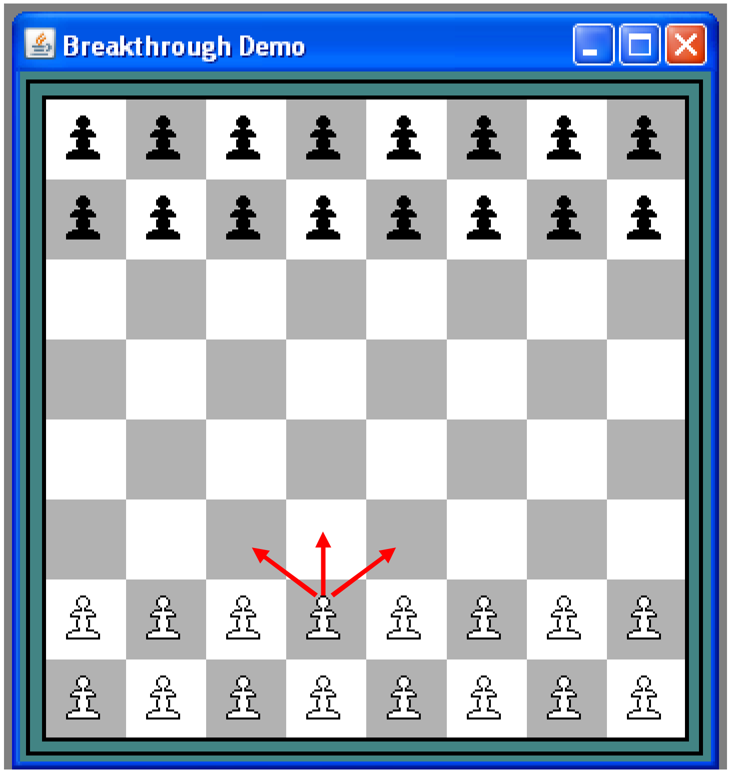
Java vs. ArchDoc

- Java Perspective
  - Package Explorer
  - File/Java Search
  - Type Hierarchy
  - Call Hierarchy
  - Class Diagrams

- ArchDoc
  - Abstract Object Tree
  - Abstract Object Search
  - Related Objects and Edges
  - Abstract Stack
  - Partial Graph View

Task on MiniDraw (MD)

- Validate piece movements on board game



- Pieces move to non-empty squares only
- Direction of movement is straight or diagonal
- Direction of movement has to be diagonal if capture of opponent pieces

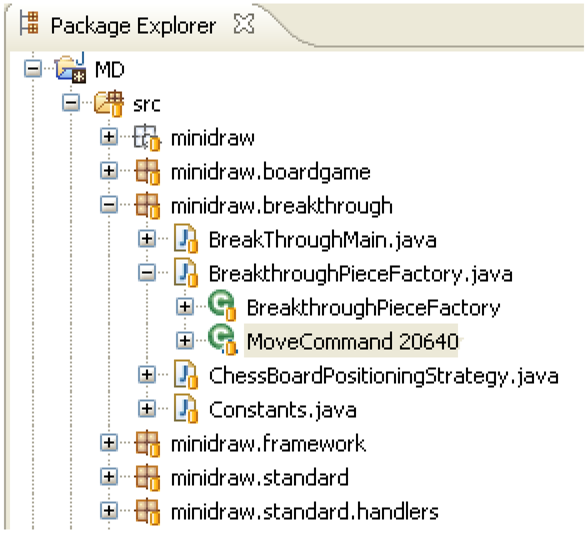
Developer Questions

- Q1: What are the architectural tiers?
- Q2: Where is canonical object of type A created?
- Q3: Can one retrieve related types/objects of type A?
- Q4: Can one access a canonical object of type A from type B?
- Q5: What are the concrete types of a canonical object of type A at runtime?

Q1: What are the architectural tiers?

- Hierarchy of classes organized into packages
- Packages, classes or interfaces sorted alphabetically
- Tiers not visible in Java code

Eclipse Package Explorer



```
MD
  src
    minidraw
    minidraw.boardgame
    minidraw.breakthrough
      BreakThroughMain.java
      BreakthroughPieceFactory.java
        BreakthroughPieceFactory
        MoveCommand 20640
      ChessBoardPositioningStrategy.java
      Constants.java
    minidraw.framework
    minidraw.standard
    minidraw.standard.handlers
```

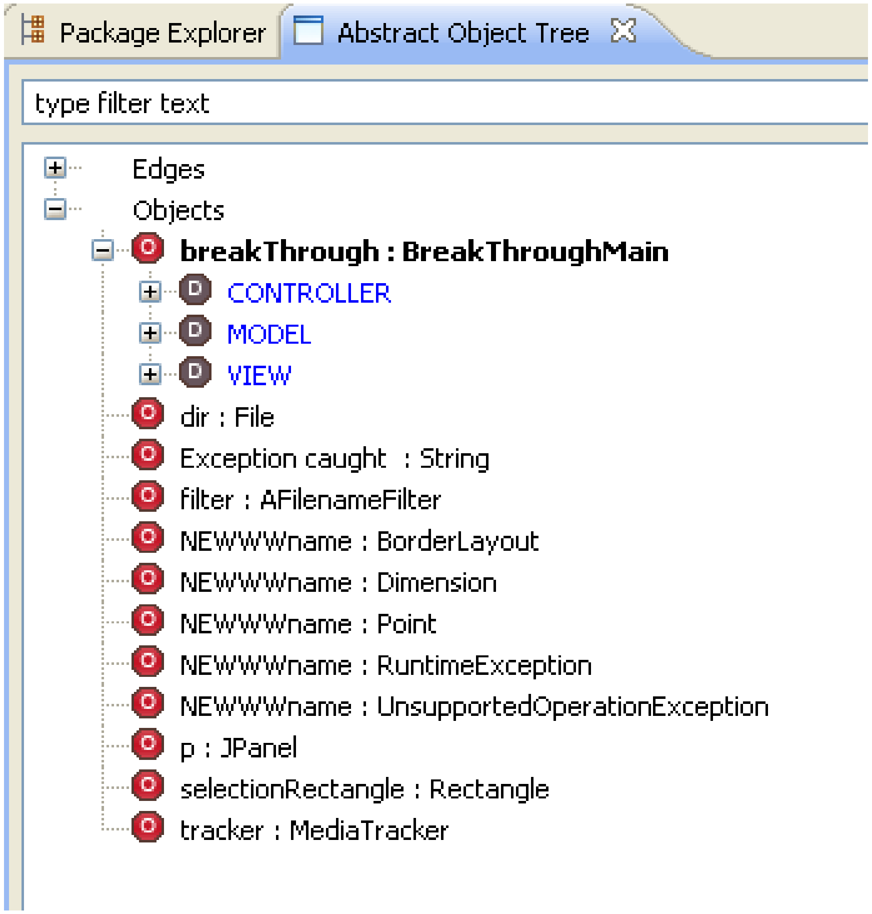
Legend

📦 Packages

Q1: What are the architectural tiers?

- Hierarchy of abstract objects and domains
- Top-level domains are the architectural tiers

ArchDoc Abstract Object Tree



- Edges
- Objects
  - O breakThrough : BreakThroughMain
    - D CONTROLLER
    - D MODEL
    - D VIEW
    - O dir : File
    - O Exception caught : String
    - O filter : AFilenameFilter
    - O NEWWWname : BorderLayout
    - O NEWWWname : Dimension
    - O NEWWWname : Point
    - O NEWWWname : RuntimeException
    - O NEWWWname : UnsupportedOperationException
    - O p : JPanel
    - O selectionRectangle : Rectangle
    - O tracker : MediaTracker

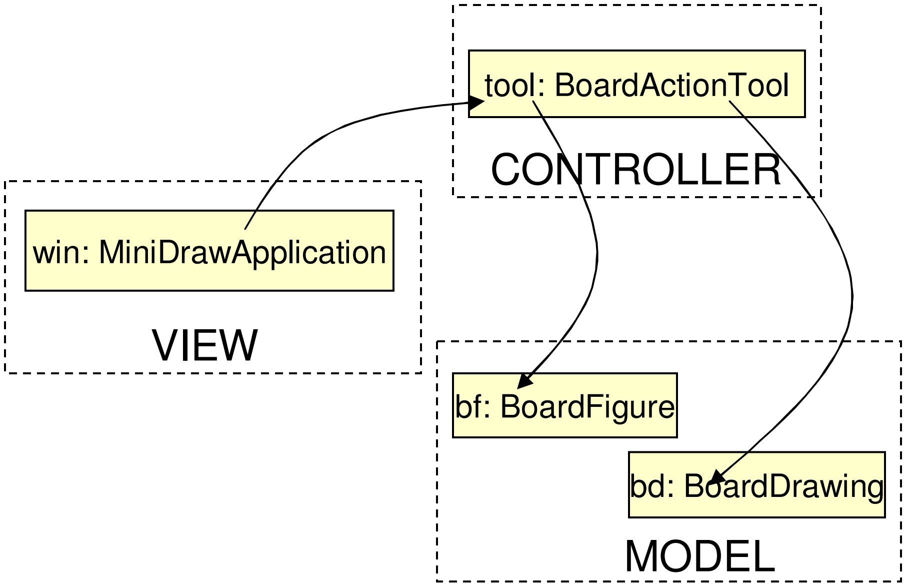
Legend

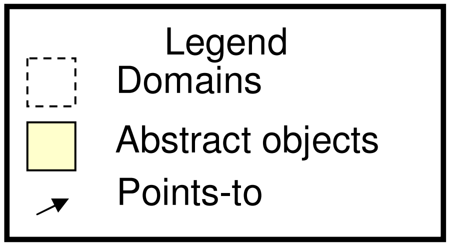
D Domains
O Abstract objects

Q1: What are the architectural tiers?

ArchDoc Hierarchical Abstract Object Graph

- Shows that MD follows 3-tiered style
- Developers can focus on objects in CONTROLLER or MODEL for this task



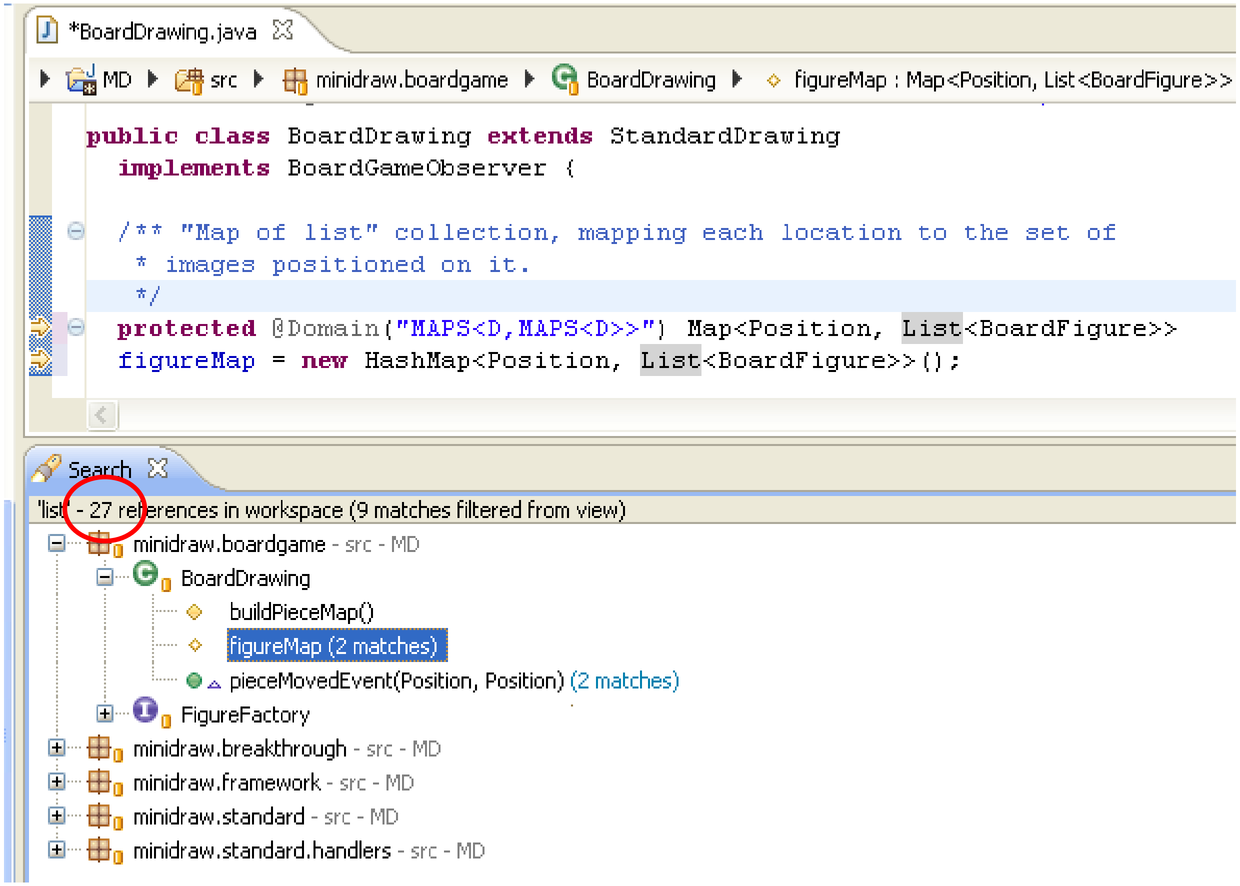


Q2: Where is canonical object of type A created?

Eclipse Java Search

Shows results from

- Comments
- Declaration points
- Usage points
- Java libraries


*BoardDrawing.java
MD ▸ src ▸ minidraw.boardgame ▸ BoardDrawing ▸ figureMap : Map<Position, List<BoardFigure>>
Search

```
public class BoardDrawing extends StandardDrawing
  implements BoardGameObserver {

  /** "Map of list" collection, mapping each location to the set of
   * images positioned on it.
   */
  protected @Domain("MAPS<D,MAPS<D>>") Map<Position, List<BoardFigure>>
  figureMap = new HashMap<Position, List<BoardFigure>>();
```

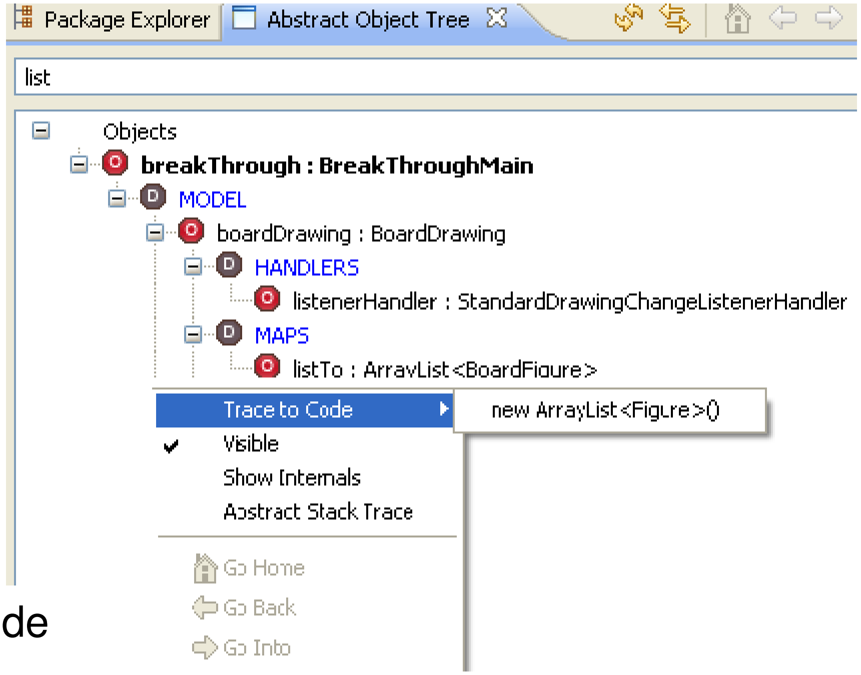
'list' - 27 references in workspace (9 matches filtered from view)

```
⊟ minidraw.boardgame - src - MD
   ⊟ BoardDrawing
        buildPieceMap()
        figureMap (2 matches)
        pieceMovedEvent(Position, Position) (2 matches)
   ⊞ FigureFactory
⊞ minidraw.breakthrough - src - MD
⊞ minidraw.framework - src - MD
⊞ minidraw.standard - src - MD
⊞ minidraw.standard.handlers - src - MD
```

Q2: Where is canonical object of type A created?

- Search for abstract objects by type, name
- Trace back to object creation expressions

ArchDoc Object Search



```
list
Objects
  breakThrough : BreakThroughMain
    MODEL
      boardDrawing : BoardDrawing
        HANDLERS
          listenerHandler : StandardDrawingChangeListenerHandler
        MAPS
          listTo : ArrayList<BoardFigure>
new ArrayList<Figure>()
```

Trace to code

```
@Domains({"owned", "MAPS"})
@DomainParams({"U","L","D"})
@DomainInherits({"BoardGameObserver <U,L,D>"})
class BoardDrawing implements BoardGameObserver {
 void pieceMovEvent(@Domain("D") Pos from, @Domain("D") Pos to) {
  @Domain("MAPS<D<U,L,D>>")
  List<BoardFigure> listTo = new ArrayList<BoardFigure>();
 }
}
```

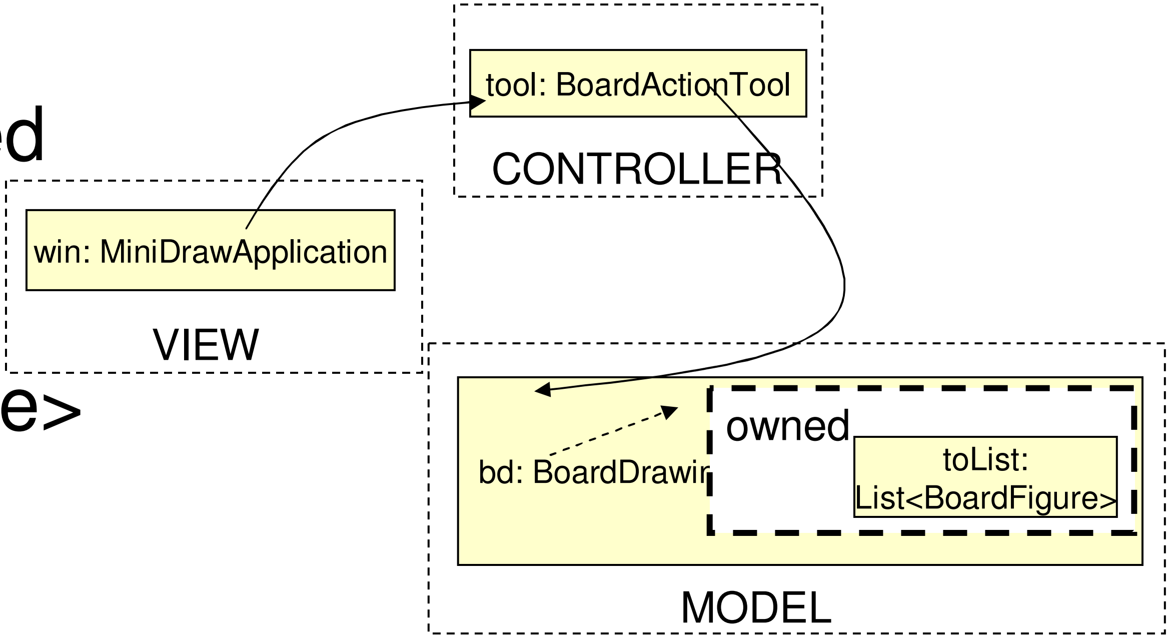
Legend

D	Domains
O	Abstract objects

Q2: Where is canonical object of type A created?

- Instances of type BoardFigure is created in BoardDrawing
- Board pieces are ArrayList<BoardFigure>

ArchDoc Hierarchical Abstract Object Graph



Legend

⬚ Domains

🟨 Abstract objects

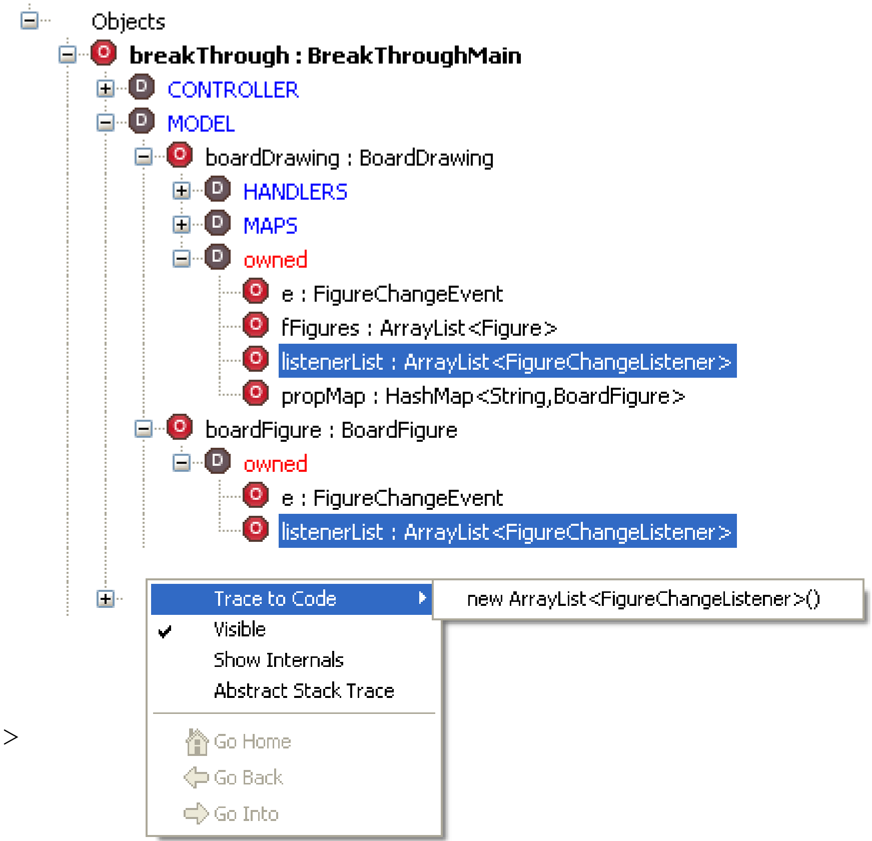
↗ Points-to

Does abstract runtime structure differ from the code structure?

E.g., do abstract objects of the same type appear in different parts of the object tree?

[Abi-Antoun et al., SCAM, 2014]

ArchDoc Abstract Object Tree



```
Objects
  breakThrough : BreakThroughMain
    CONTROLLER
    MODEL
      boardDrawing : BoardDrawing
        HANDLERS
        MAPS
        owned
          e : FigureChangeEvent
          fFigures : ArrayList<Figure>
          listenerList : ArrayList<FigureChangeListener>
          propMap : HashMap<String,BoardFigure>
      boardFigure : BoardFigure
        owned
          e : FigureChangeEvent
          listenerList : ArrayList<FigureChangeListener>
```

Trace to code

```
@Domains({"owned"})
@DomainParams({"U","L","D"})
@DomainInherits({"Figure<U,L,D>"})
abstract class AbstractFigure implements Figure {
 @Domain("owned<D<U,L,D>>") List<FigureChangeListener>
 listenerList;
 public AbstractFigure() {
  listenerList = new ArrayList<FigureChangeListener>();
 }
}
```

Legend

D Domains

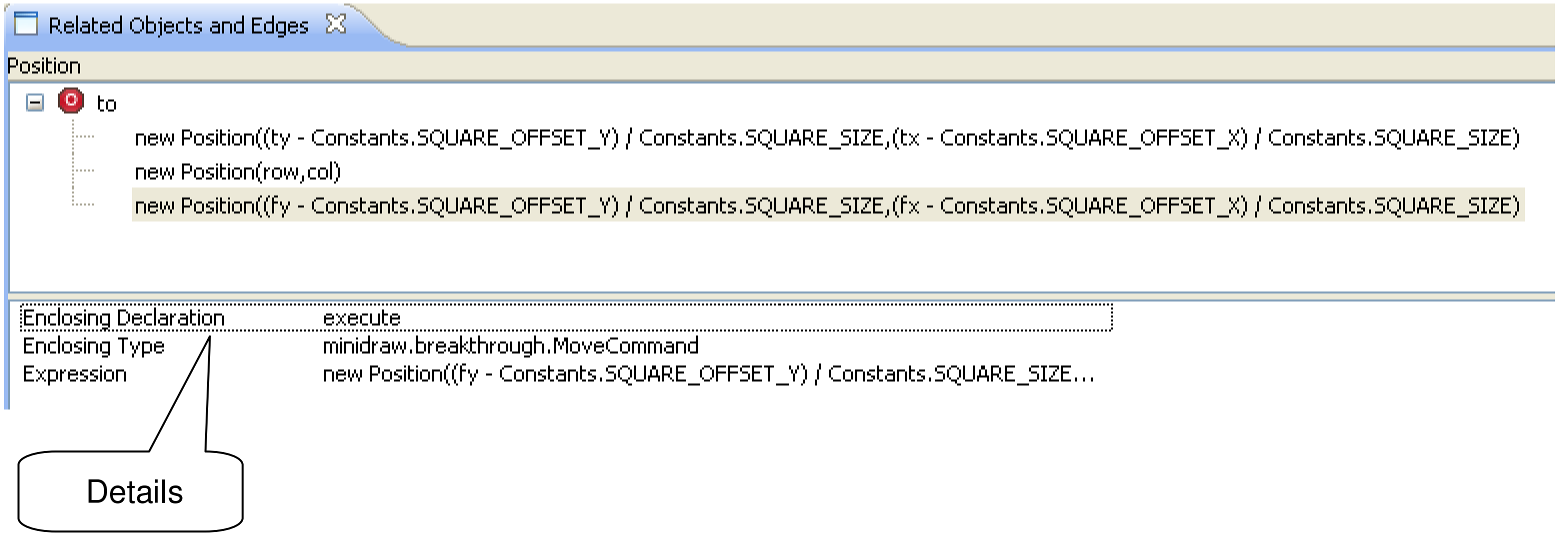
O Abstract objects

Q3: Can one retrieve related types/objects of type A?

Code Fragment

```
@Domains({"owned" })
@DomainParams({"U","L","D"})
@DomainInherits({"Command<U,L,D>"})
class MoveCommand implements Command {
 @Domain("D") Position VIRTUAL_from = Null;
 ...
}
```

- No related Java perspective functionality
- Enclosing type, declaration are displayed as details



Position

to

new Position((ty - Constants.SQUARE_OFFSET_Y) / Constants.SQUARE_SIZE,(tx - Constants.SQUARE_OFFSET_X) / Constants.SQUARE_SIZE)

new Position(row,col)

new Position((fy - Constants.SQUARE_OFFSET_Y) / Constants.SQUARE_SIZE,(fx - Constants.SQUARE_OFFSET_X) / Constants.SQUARE_SIZE)

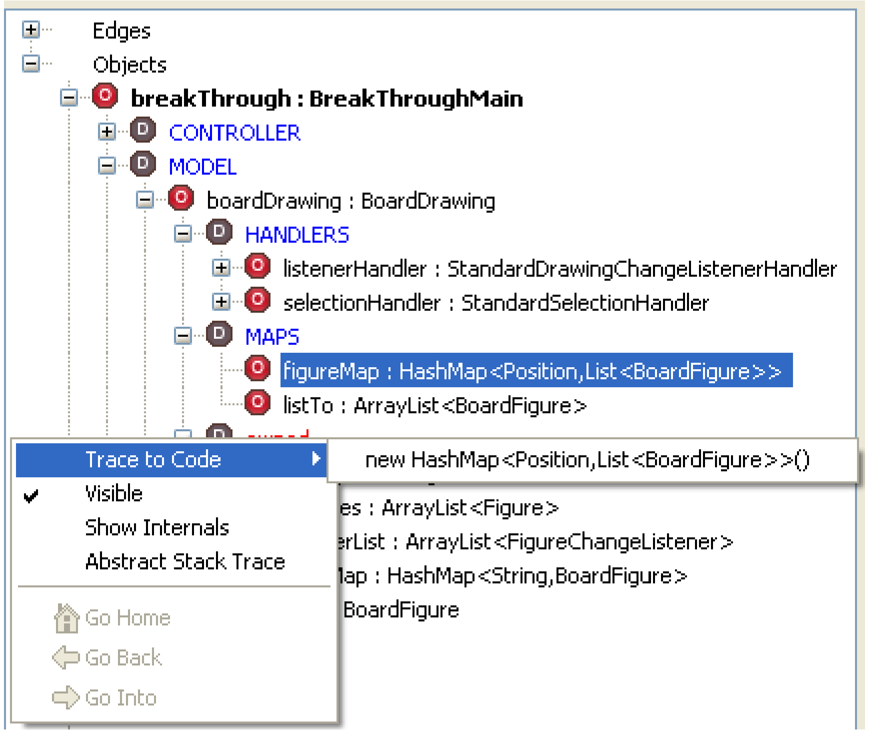
Enclosing Declaration	execute
Enclosing Type	minidraw.breakthrough.MoveCommand
Expression	new Position((fy - Constants.SQUARE_OFFSET_Y) / Constants.SQUARE_SIZE...

Details

Q4: Can one access a canonical object of type A from type B?

- Hierarchy in abstract object tree
- Objects representing data structures are beneath relevant objects

ArchDoc Abstract Object Tree



```
⊞ Edges
⊟ Objects
  ⊟ O breakThrough : BreakThroughMain
    ⊞ D CONTROLLER
    ⊟ D MODEL
      ⊟ O boardDrawing : BoardDrawing
        ⊟ D HANDLERS
          ⊞ O listenerHandler : StandardDrawingChangeListenerHandler
          ⊞ O selectionHandler : StandardSelectionHandler
        ⊟ D MAPS
            O figureMap : HashMap<Position,List<BoardFigure>>
            O listTo : ArrayList<BoardFigure>
        ⊟ D owned
            es : ArrayList<Figure>
            erList : ArrayList<FigureChangeListener>
            lap : HashMap<String,BoardFigure>
            BoardFigure
```

new HashMap<Position,List<BoardFigure>>()

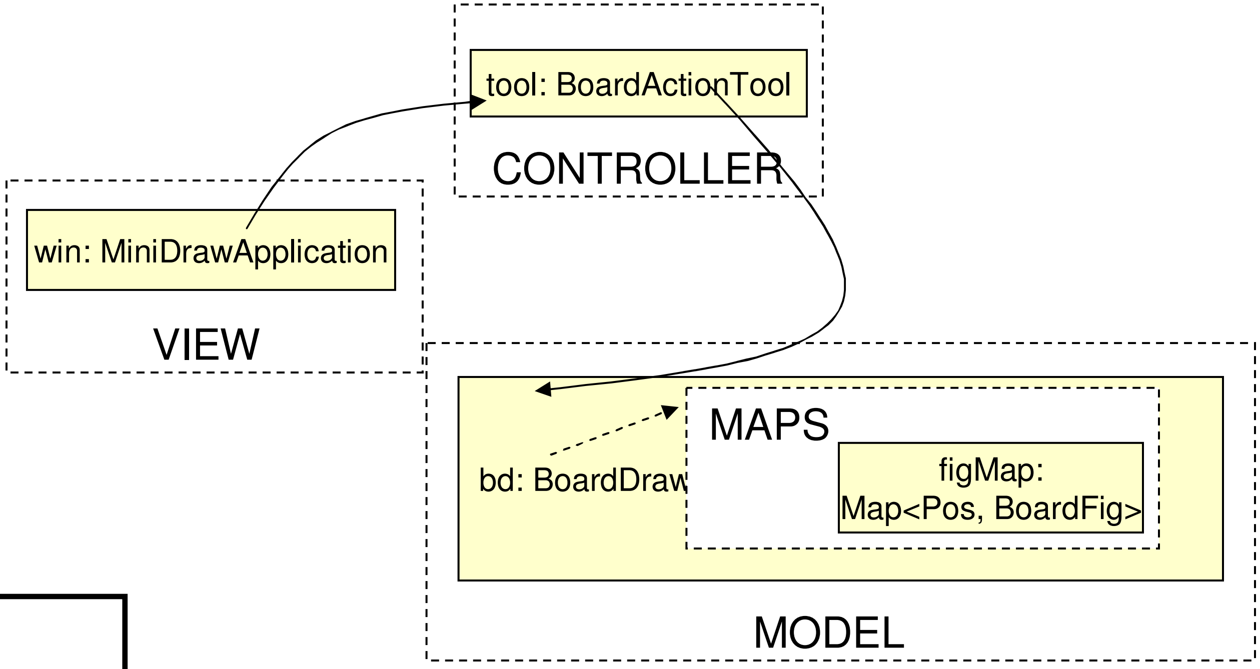
Legend

D Domains

O Abstract objects

Q4: Can one access a canonical object of type A from type B?

ArchDoc Hierarchical Abstract Object Graph



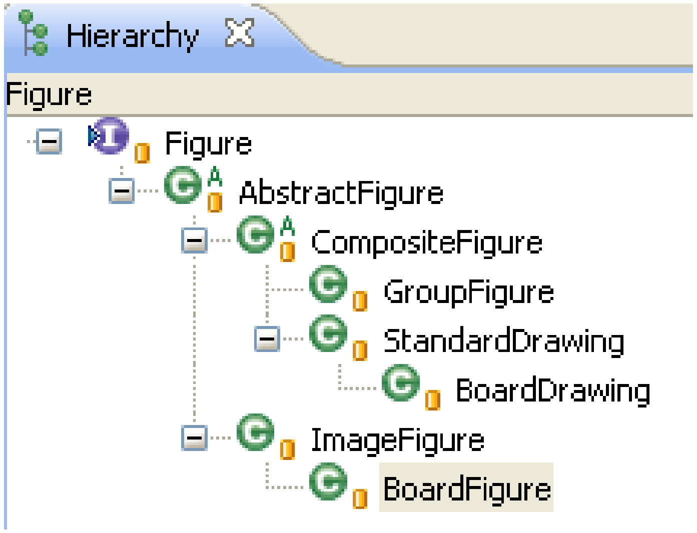
Legend

- ⬚ Domains
- 🟨 Abstract objects
- ↗ Points-to

Q5: What are the concrete types of a canonical object of type A at runtime?

- Shows all possible subtypes of Figure
- Including non concrete types

Eclipse Type Hierarchy

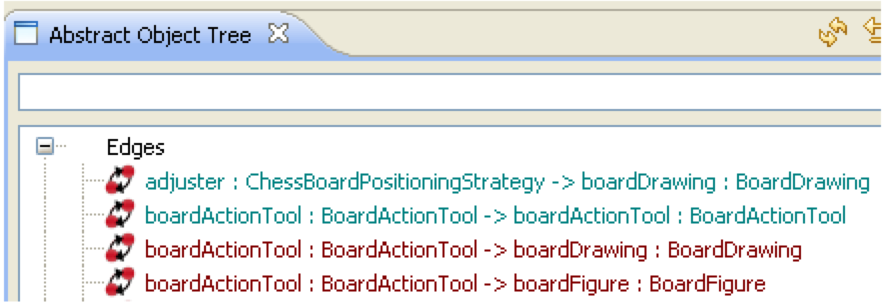


Figure

```
Figure
  AbstractFigure
    CompositeFigure
      GroupFigure
      StandardDrawing
        BoardDrawing
    ImageFigure
      BoardFigure
```

Q5: What are the concrete types of a canonical object of type A at runtime?

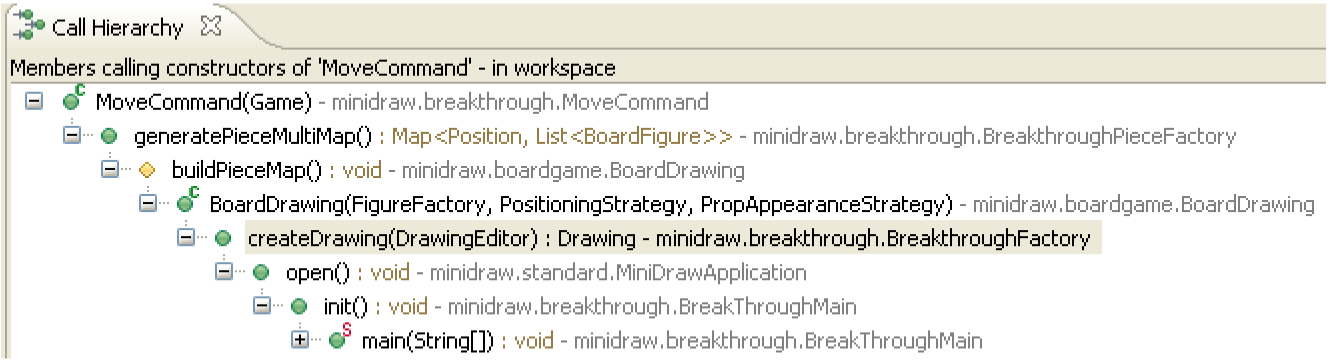
ArchDoc Abstract Object Tree



Edges
- adjuster : ChessBoardPositioningStrategy -> boardDrawing : BoardDrawing
- boardActionTool : BoardActionTool -> boardActionTool : BoardActionTool
- boardActionTool : BoardActionTool -> boardDrawing : BoardDrawing
- boardActionTool : BoardActionTool -> boardFigure : BoardFigure

- Interested only in concrete types of Figure declared in BoardActionTool
- Refers to 2 points-to edges from BoardActionTool

Eclipse Call Hierarchy



Call Hierarchy

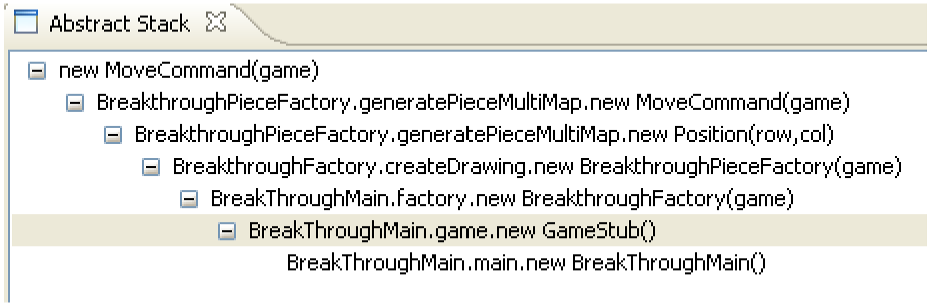
Members calling constructors of 'MoveCommand' - in workspace

```
⊟ MoveCommand(Game) - minidraw.breakthrough.MoveCommand
  ⊟ generatePieceMultiMap() : Map<Position, List<BoardFigure>> - minidraw.breakthrough.BreakthroughPieceFactory
    ⊟ buildPieceMap() : void - minidraw.boardgame.BoardDrawing
      ⊟ BoardDrawing(FigureFactory, PositioningStrategy, PropAppearanceStrategy) - minidraw.boardgame.BoardDrawing
        ⊟ createDrawing(DrawingEditor) : Drawing - minidraw.breakthrough.BreakthroughFactory
          ⊟ open() : void - minidraw.standard.MiniDrawApplication
            ⊟ init() : void - minidraw.breakthrough.BreakThroughMain
              ⊞ main(String[]) : void - minidraw.breakthrough.BreakThroughMain
```

- Shows caller and callees transitively for a selected method
- Traces to method invocations

ArchDoc Abstract Stack

- Contrast with Call Hierarchy
- Shows object creation hierarchy
- Shows abstract interpretation contexts
- Exposes notion of "object sensitivity"



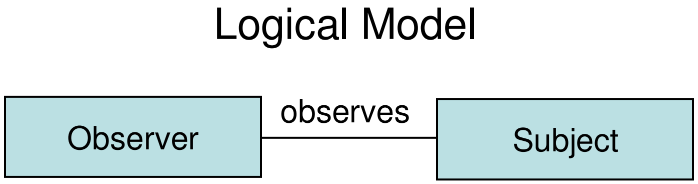
```
⊟ new MoveCommand(game)
    ⊟ BreakthroughPieceFactory.generatePieceMultiMap.new MoveCommand(game)
        ⊟ BreakthroughPieceFactory.generatePieceMultiMap.new Position(row,col)
            ⊟ BreakthroughFactory.createDrawing.new BreakthroughPieceFactory(game)
                ⊟ BreakThroughMain.factory.new BreakthroughFactory(game)
                    ⊟ BreakThroughMain.game.new GameStub()
                          BreakThroughMain.main.new BreakThroughMain()
```

Applications

- Explaining Design Patterns
- Explaining Shallow vs. Deep Copy

Explaining Design Patterns

- Observer design pattern.



- Logical model applicable to MD
  - Observer: BoardDrawing
  - Subject: GameStub

Explaining Design Patterns

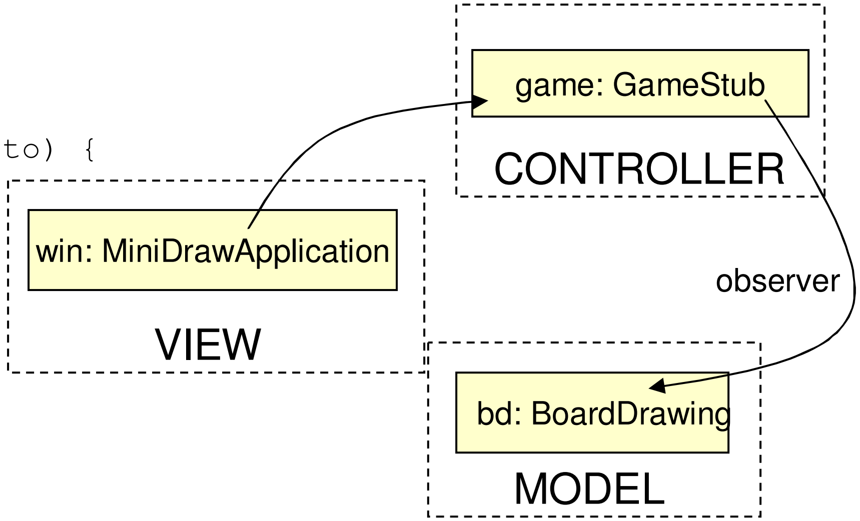
- Observer design pattern.

Code Fragment

```
@Domains({"owned"})
@DomainParams({"U","L","D"})
@DomainInherits({"Game<U,L,D>"})
class GameStub implements Game {
 @Domain("D<U,L,D>") BoardGameObserver observer;
 void move(@Domain("D") Pos from, @Domain("D") Pos to) {
 observer.pieceMovedEvent(from, to);
 }
}
```

BoardDrawing <: BoardGameObserver

ArchDoc Hierarchical Abstract Object Graph



Legend

⬚ Domains

🟨 Abstract objects

↗ Points-to

Related tools

- Objektgraph: flat and non abstract depictions of runtime structure
  [Buck et al., SPLASH, 2013]
- Code exploration: focus is on code structure
  [Kollman et al., WCRE, 2002]
- Call graph: focus is on visualizing call graphs
  [Bohnet and D¨ollner, WODA, 2006]
- Heap exploration: focus is on visualize and interactively explore snapshots of the heap
  [Kelley et al., Information Visualization, 2012]

Conclusion

- ArchDoc complements existing design-time perspective in Eclipse
- ArchDoc helps answer developer questions based on abstract runtime structure

Future work

- Evaluate the tool in user studies
  - Replicate results from previous experiment

  [Ammar and Abi-Antoun, WCRE, 2012]
- Use the tool in educational setting
  - Beginners learning design patterns, etc.
  - Use in laboratory component of course