Abstract Classes and Interfaces

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688R, Spring Semester 2023

March 2, 2023

Overview

Part 3

Quick Review

Working with Abstract Classes

Abstract Classes

Abstract classes provide an abstract view of a real-world entity or concept. They are an ideal mechanism when you want to create something for objects that are closely related in a hierarchy.

Implementation

- An abstract class is a class that is declared abstract. It may or may not include abstract methods.
- You cannot create an object from an abstract class – but they can be sub-classed.
- The subclasses will usually provide implementations for all of the abstract methods in its parent class.

Programming to an Interface

Motivation

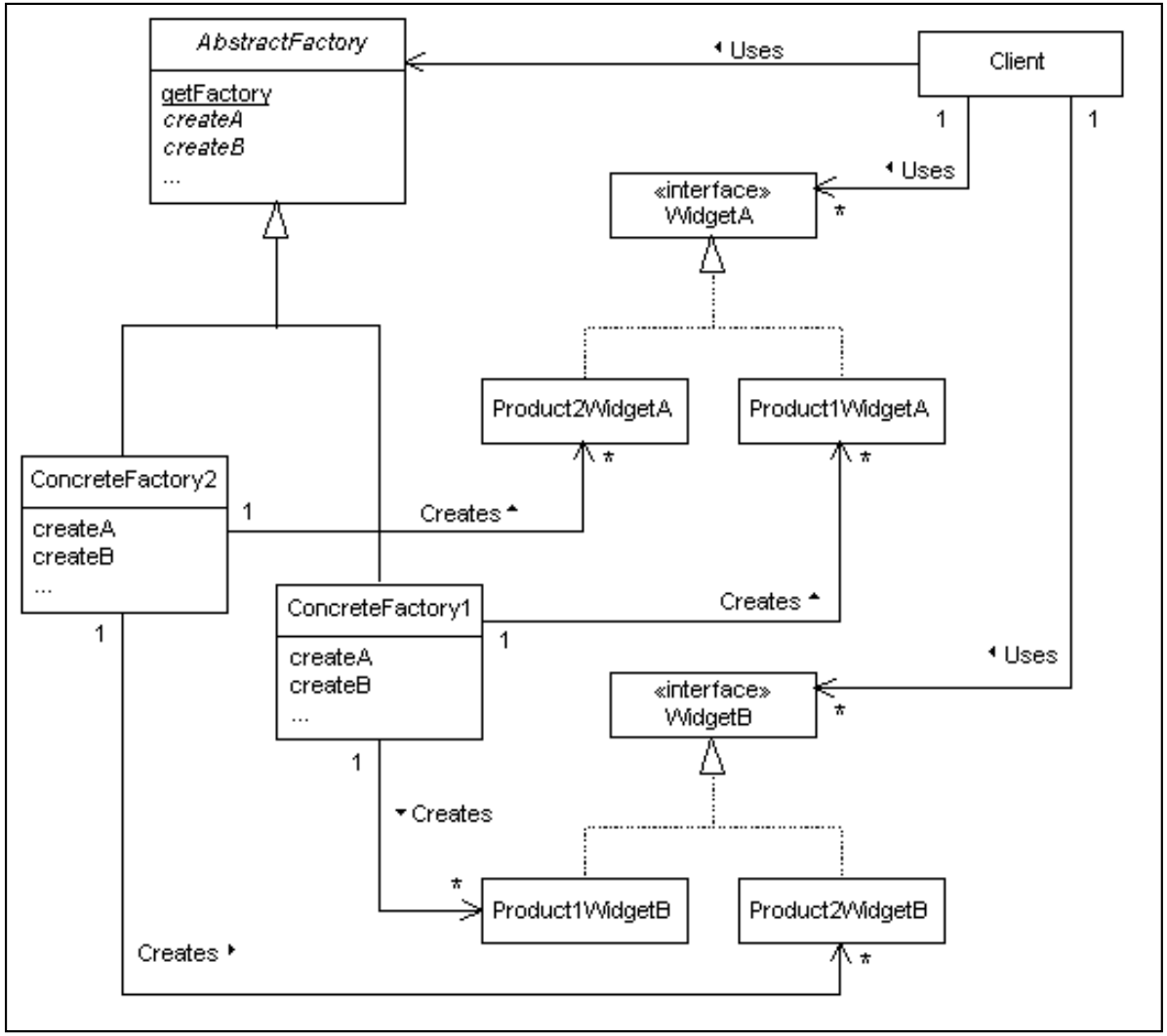
- Interfaces are the mechanism by which components describe what they do, but not how they do it.
- Interface abstractions are appropriate for collections of objects that provide common functionality, but are otherwise unrelated.

Implementation

- An interface defines a set of methods without providing an implementation for them.
- An interface does not have a constructor – therefore, it cannot be instantiated as a concrete object.
- Any concrete class the implements the interface must provide implementations for all of the methods listed in the interface.

Five Applications

Application 1. Two Factories making Widgets



Application 1. Two Factories making Widgets

Points to Note:

- The client works with an abstract model of a factory and two types of widgets, A and B, but only knows about their interfaces.
- The interfaces separate the client from details of how A and B are manufactured.
- Thus, a factory can change and the client will be completely unaware.

Application 2. Parsing/Evaluation of Functions with JEval

Purpose:

- JEval parses and evaluates dynamic and static expressions at run time.
- As such, it is a great solution for filtering streams of data at runtime.

Features:

- Supports mathematical, Boolean, String and functional expressions.
- Supports all major mathematical and Boolean operators.
- Supports custom functions.
- 39 Math and String functions built in and ready to use.
- Supports variables and nested functions.

Application 2. Evaluation of Functions with JEval

Examples: Relational and Arithmetic Expressions

- String sExp = "(2 < 3) || ((1 == 1) && (3 < 3))";
- String sExp = "1 + 2 + 3*4 + 10.0/2.5";
- String sExp = "1 + abs(-1)";
- String sExp = "atan2(atan2(1, 1), 1)";
- String sExp = "acos(-1.0)";

Examples: Working with Strings

- String sExp = "toLowerCase('Hello World!')";
- String sExp = "toUpperCase(trim(trim(' a b c ')))";

Application 2. Evaluation of Functions with JEval

Examples: Working with variables

```
String sEexp = "#{a} >= 2 && #{b} >= 5 && #{c} >= 8";

Long a = (Long) row.get(0);
evaluator.putVariable("a", a.toString());
Long b = (Long) row.get(1);
evaluator.putVariable("b", a.toString());
Long c = (Long) row.get(2);
evaluator.putVariable("c", a.toString());

... etc ...

String result01 = evaluator.evaluate( sExp );
```

Application 2. Evaluation of Functions with JEval

Builtin String Functions

CharAt.java	CompareTo.java	Concat.java
EndsWith.java	Equals.java	Eval.java
IndexOf.java	LastIndexOf.java	Length.java
Replace.java	StartsWith.java	Substring.java
ToLowerCase.java	ToUpperCase.java	Trim.java

Builtin Math Functions

Abs.java	Acos.java	Asin.java
Atan.java	Atan2.java	Ceil.java
Cos.java	Exp.java	Floor.java
Log.java	Max.java	Min.java
Pow.java	Random.java	Rint.java
Round.java	Sin.java	Sqrt.java
Tan.java	ToDegrees.java	
ToRadians.java		

Application 2. Evaluation of Functions with JEval

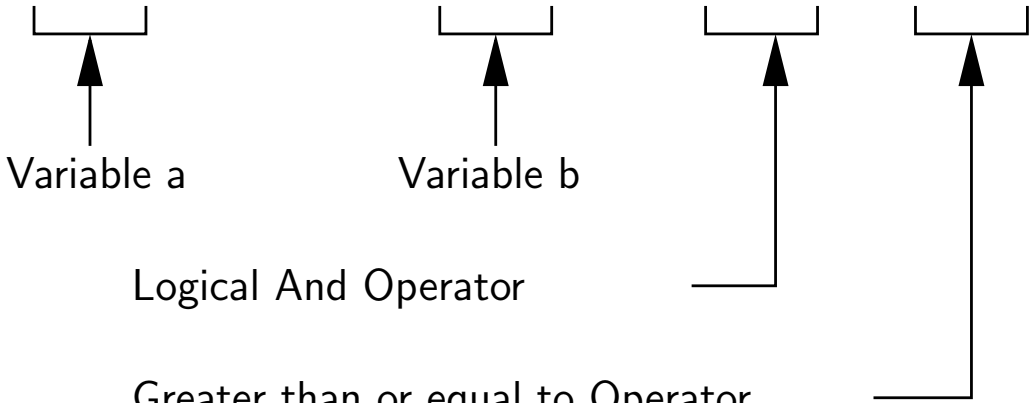
Builtin Operator Functions:

AbstractOperator.java
AdditionOperator.java
BooleanAndOperator.java
BooleanNotOperator.java
BooleanOrOperator.java
ClosedParenthesesOperator.java
DivisionOperator.java
EqualOperator.java
GreaterThanOperator.java
GreaterThanOrEqualOperator.java
LessThanOperator.java
LessThanOrEqualOperator.java
ModulusOperator.java
MultiplicationOperator.java
NotEqualOperator.java
OpenParenthesesOperator.java
Operator.java
SubtractionOperator.java

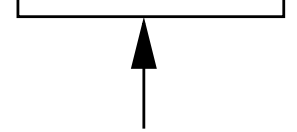
Application 2. Evaluation of Functions with JEval

Syntax and Semantics

String sEexp = "#{ a } >= 2 && #{ b } >= 6 && #{ c } >= 8";



String sEexp = " atan2 (atan2 (1, 1), 1)";



Application 2. Evaluation of Functions with JEval

Function Interface

```
public interface Function {

   // Return name of the function ...

   public String getName();

   // Execute the function for a specified argument ...

   public FunctionResult execute(Evaluator evaluator, String arguments)
}
```

Using the Function Interface

```
public class Acos implements Function { ... } ....
public class Max  implements Function { ... } ....
```

Application 2. Evaluation of Functions with JEval

Operator Interface

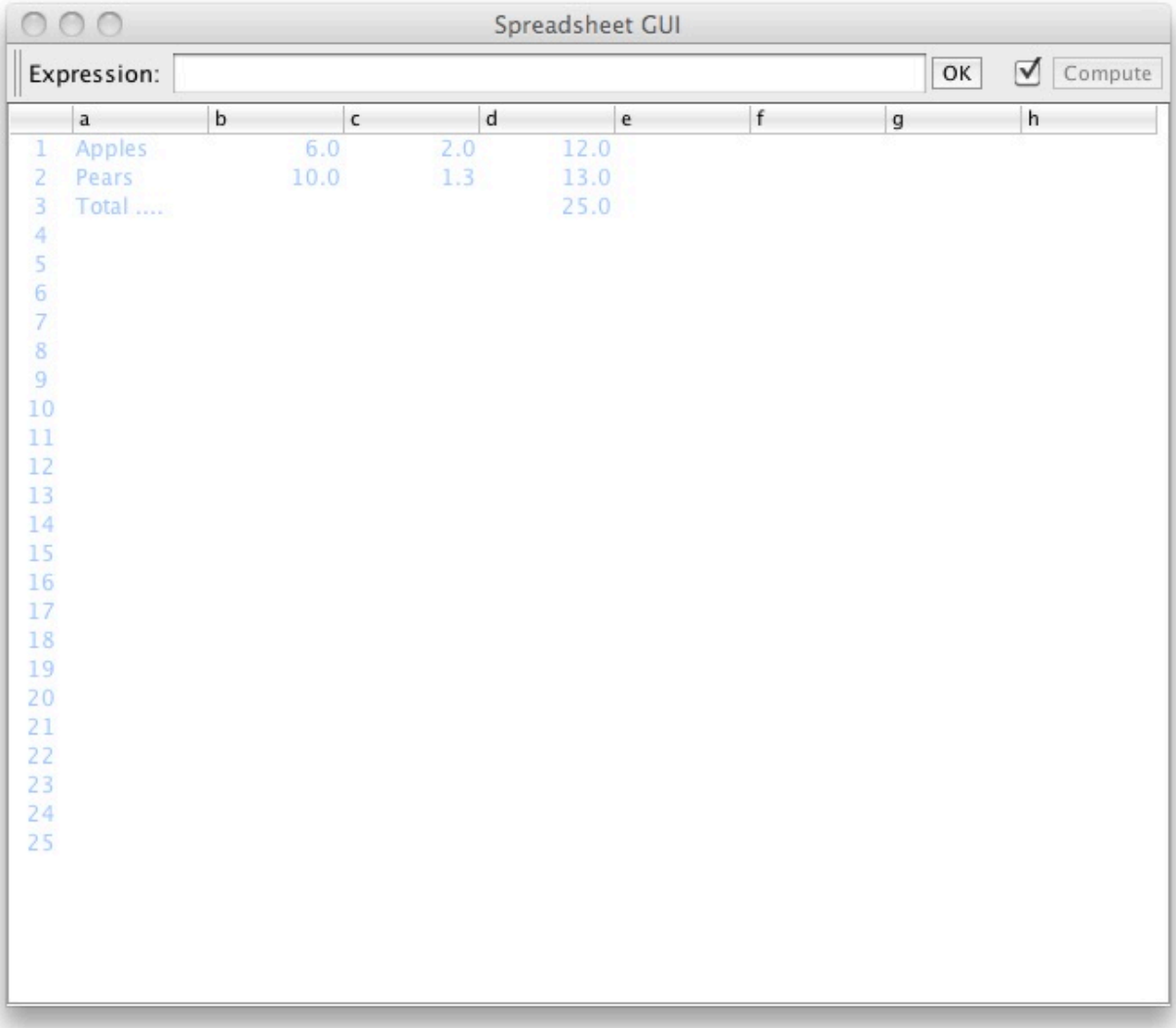
```
public interface Operator {
   // Evaluates two double operands.
   public abstract double evaluate(double  leftOperand,
                                   double rightOperand );

   // Evaluate one double operand ...
   public abstract double evaluate(final double operand);
}
```

Using the Operator Interface

```
public abstract class AbstractOperator implements Operator { ... }

public class DivisionOperator extends AbstractOperator { ... }
public class BooleanAndOperator extends AbstractOperator { ... }
```

Application 3. Using Interfaces in Spreadsheets

Application 3: Graphical Interface



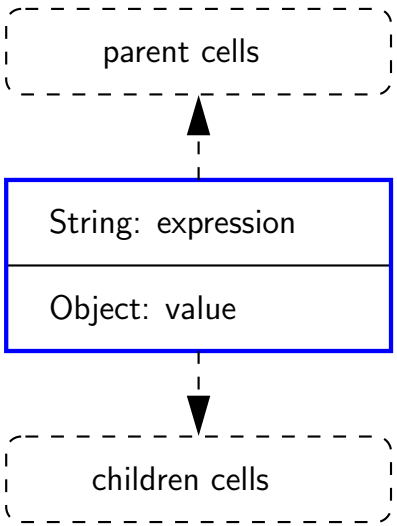
Application 3. Using Interfaces in Spreadsheets

Modeling a Spreadsheet Cell

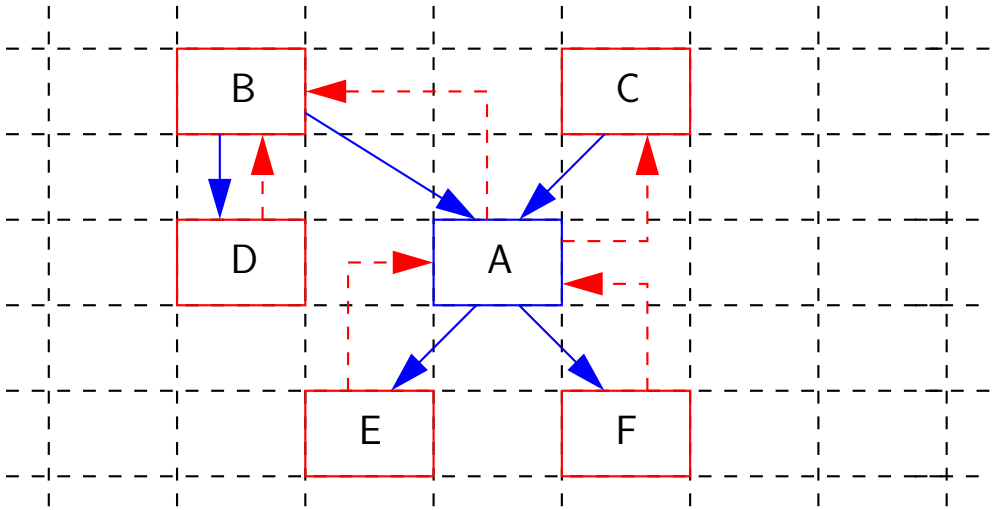
```
public class Cell {
   private String expression;    // expression in cell
   private Set<String> children; // list of cells which reference this
   private Set<String> parent;   // list of cells this references
   private Object value;         // Value of displayed cell ...

   // Class constructor

   public Cell() {
      children = new TreeSet<String>();
      parent   = new TreeSet<String>();
   }

   ..... etc .....
}
```

Application 3. Using Interfaces in Spreadsheets

Basic Cell Model



Network of Dependencies among Cells



- The parents of Cell A are cells B and C; the children are cells E and F.
- No loops in the graph of dependency relationships.
- Topological sort $\rightarrow$ update cell values in one pass.

Application 3. Using Interfaces in Spreadsheets

Basic Spreadsheet Interface

```
public interface SpreadsheetInterface {
   public static final String LOOP = "#LOOP"; // loop Error Value
   public int getColumnCount();                // Number of columns
   public int getRowCount();                   // Number of rows

   // Set and get the cell expression at prescribed location...

   public   void setExpression(String location, String expression);
   public String getExpression(String location);

   // Returns the expression stored at the cell at location.

   public Object getValue(String location);

   // Returns the value associated with the computed stored expression.

   public void recompute();
}
```

Application 3. Using Interfaces in Spreadsheets

Extended Spreadsheet Interface

```
public interface IterableSpreadsheetInterface extends SpreadsheetInterf

   // Set/get number of times to compute the value stored in each loop

   public void setMaximumIterations(int maxIterationCount);
   public  int getMaximumIterations();

   // Set/get the maximum change in value between successive loop itera

   public   void setMaximumChange(double epsilon);
   public double getMaximumChange();

   // Recompute value of all cells ...

   public void recomputeWithIteration();
}
```

Application 3. Using Interfaces in Spreadsheets

Creating the Spreadsheet Model

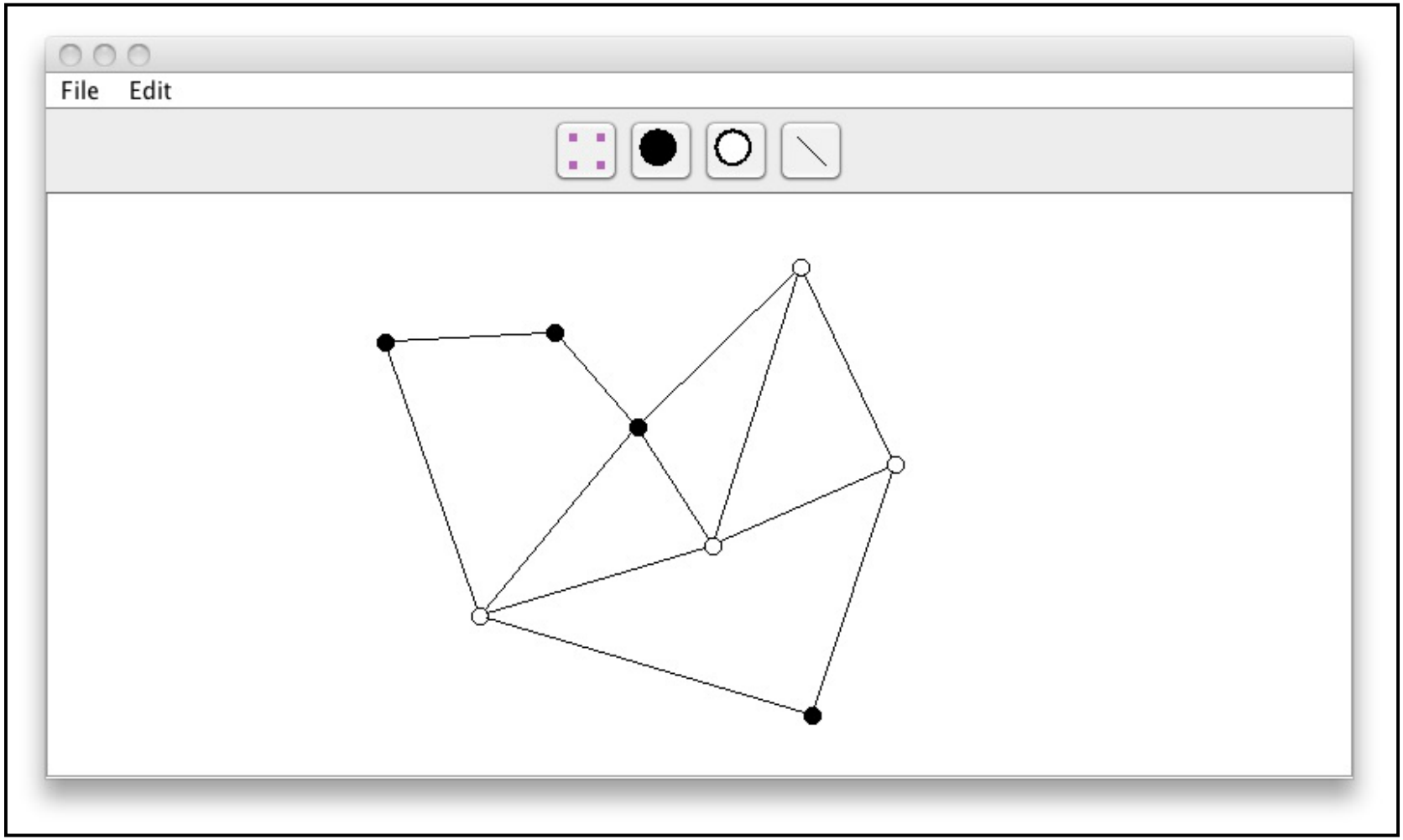
```
public class Spreadsheet implements SpreadsheetInterface {
   private int numRows, numColumns;     // no. of rows and cols
   private Map<String, Cell> cells;     // collection of all cells
   private String lastCellLocation;     // last cell accessed

   // Set expression of the cell at location ...

   public void setExpression(String location, String expression) { ...

   // Recompute value of all cells ....

   public void recompute() { ... }

   // Use DFS to check for loops in the relationships among cells ...

   private void checkLOOP(String cellLocation) { ... }
}
```

Application 3. Using Interfaces in Spreadsheets

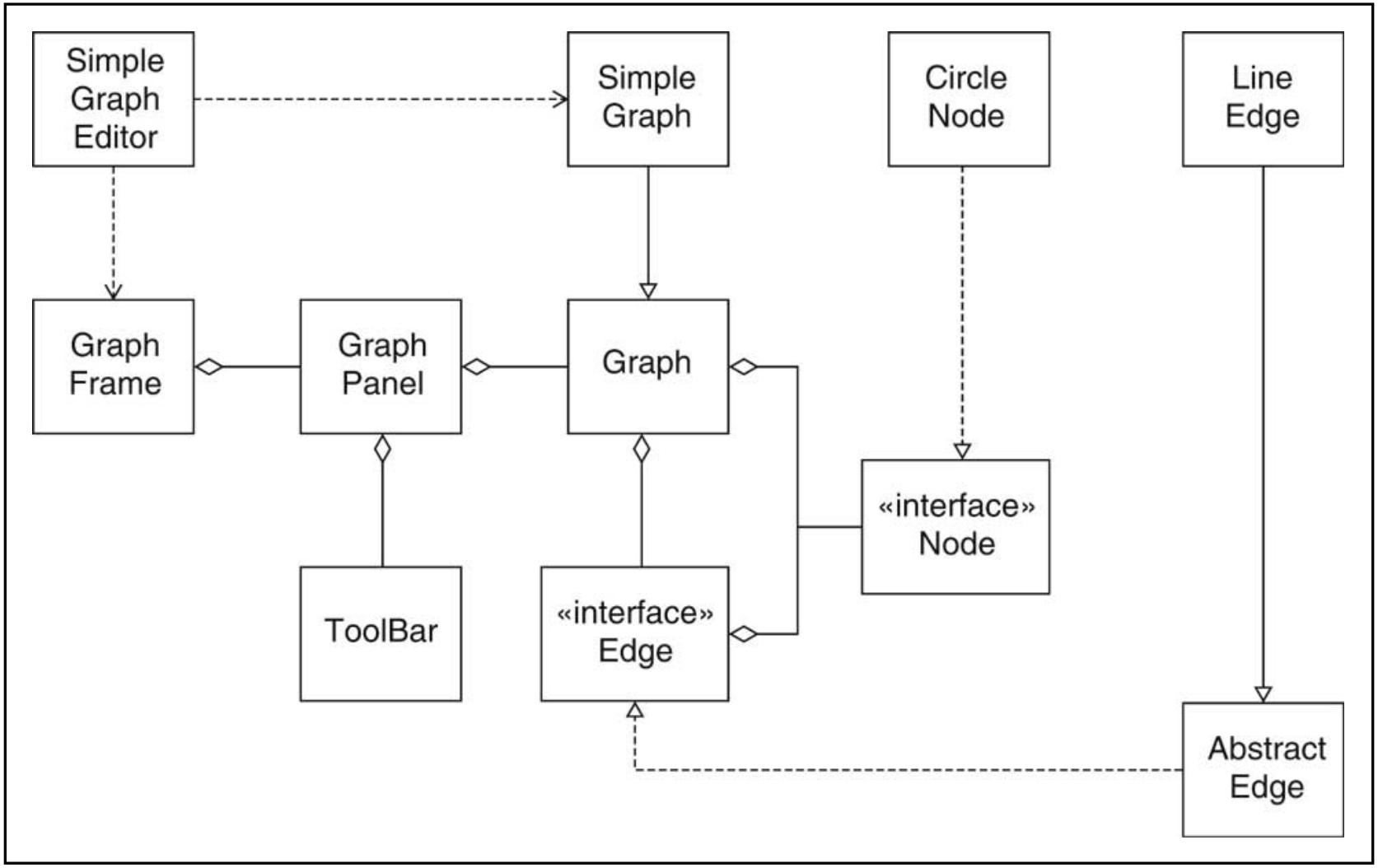
Creating a Spreadsheet Object

```
int columns = Integer.parseInt(args[0]);
int rows    = Integer.parseInt(args[1]);

SpreadsheetInterface spreadsheet = new Spreadsheet(rows, columns);

javax.swing.SwingUtilities.invokeLater(new Runnable() {
   public void run() {
      new SpreadsheetGUI("Spreadsheet GUI", spreadsheet);
   }
});
```

Application 4. Horstmann's Simple Graph Editor



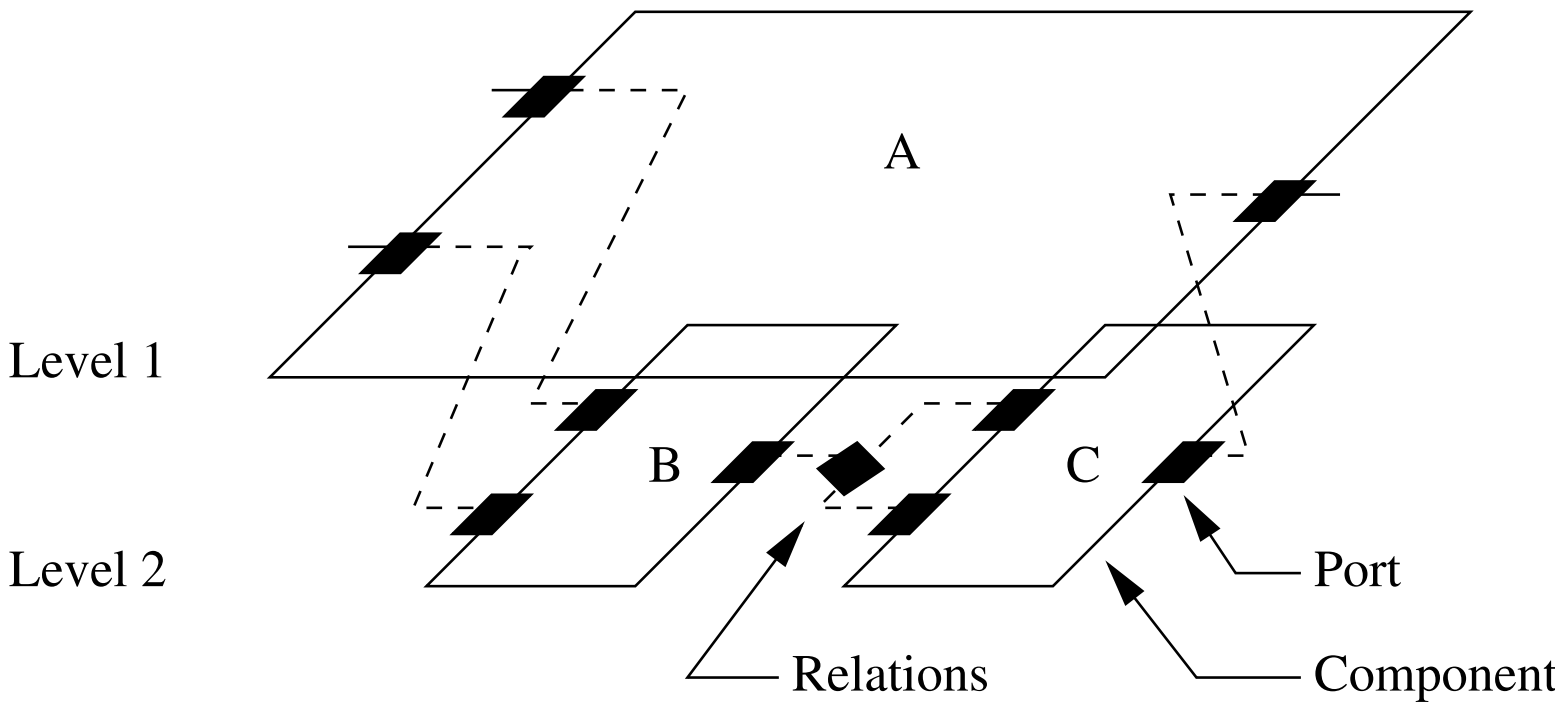
Application 4. Horstmann's Simple Graph Editor



Application 5. Architecture for Interconnect System

Problem Statement.

Hierarchy and network abstractions in a two-layer block component/container model.



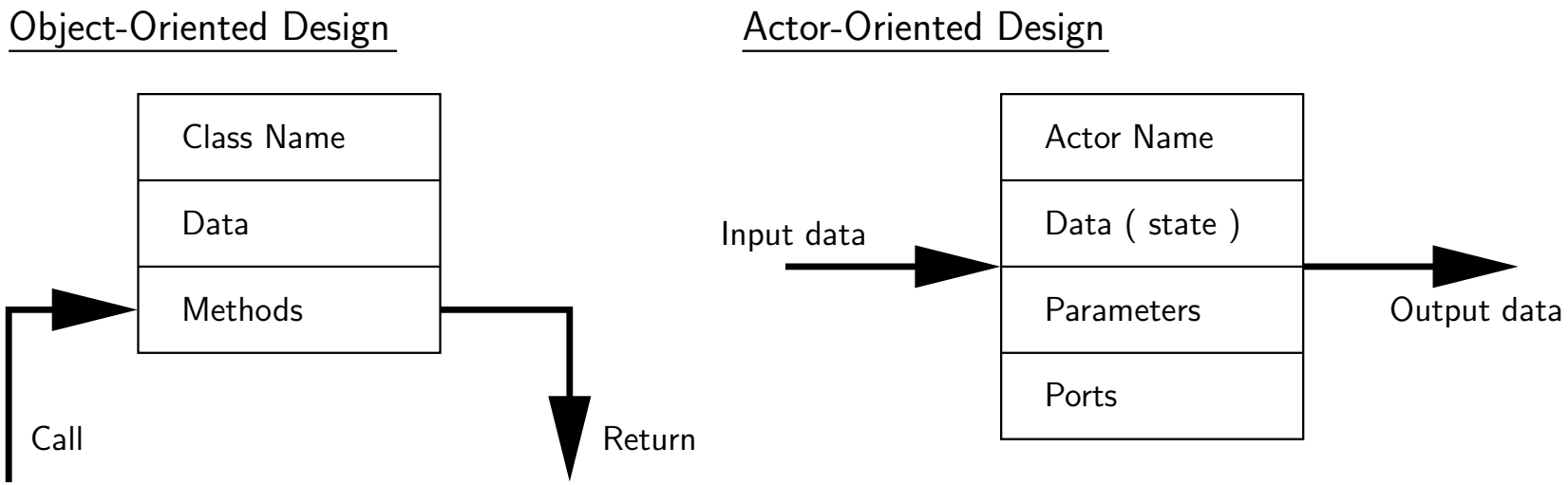
Application 5. Architecture for Interconnect System

Organizational Constraints:

- Within a hierarchy, each level is logically connected to the levels above and below it.
- A port cannot be contained by more than one entity. Links cannot cross levels in the hierarchy,
- Port-to-port communications must have compatible data types (e.g., signal, energy).

Application 5. Architecture for Interconnect System

Actor-Oriented Models and Design (adapted from Lee, 2003)



Object-Oriented Modeling and Design

- Components interact primarily through method calls (transfer of control).

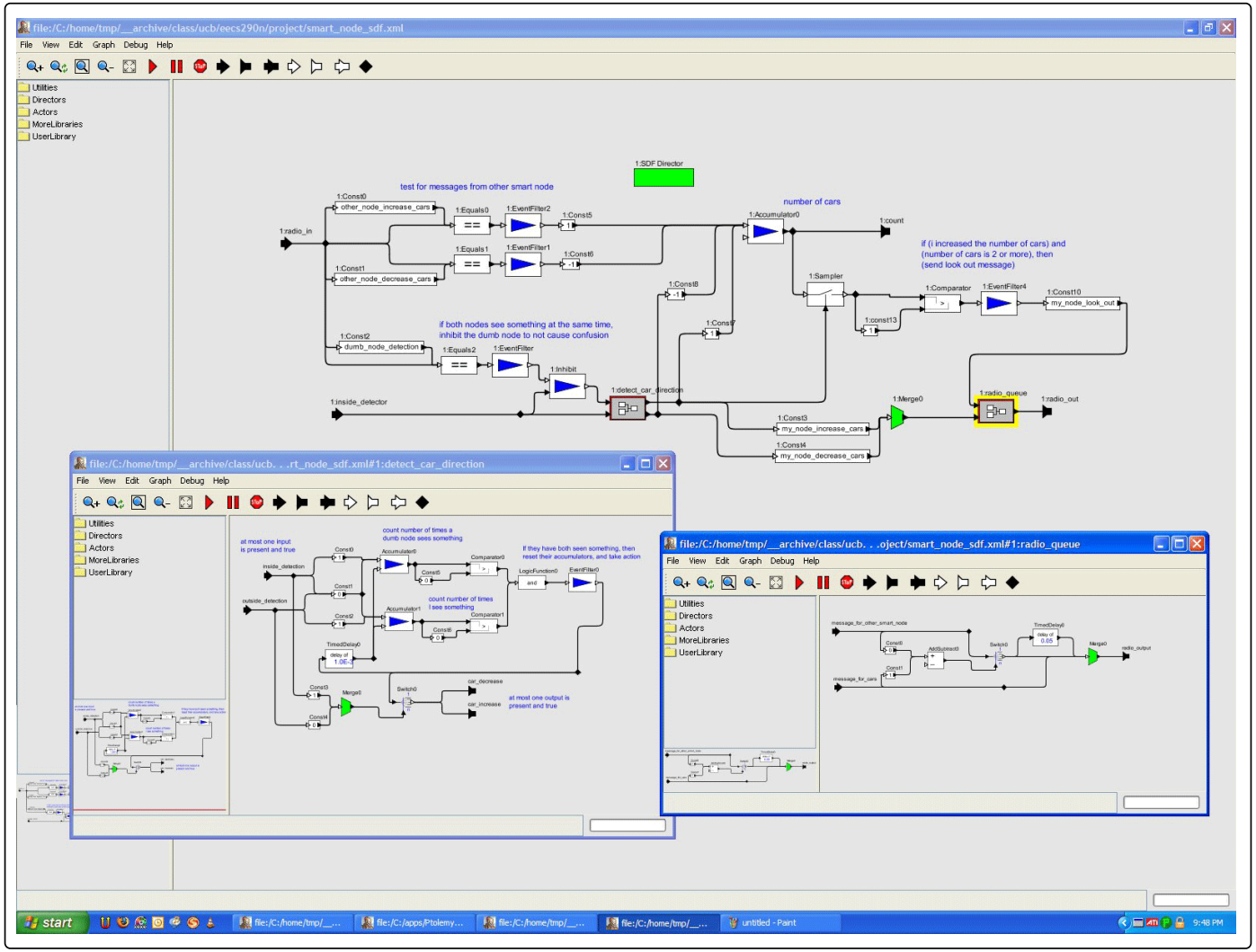
Application 5. Architecture for Interconnect System

Actor-Oriented Modeling and Design

- Components interact via some sort of messaging scheme that is typically concurrent.
- Constraints in the flow of control define the model of computation.
- Rules define what an actor does (e.g. perform external communication) and when.

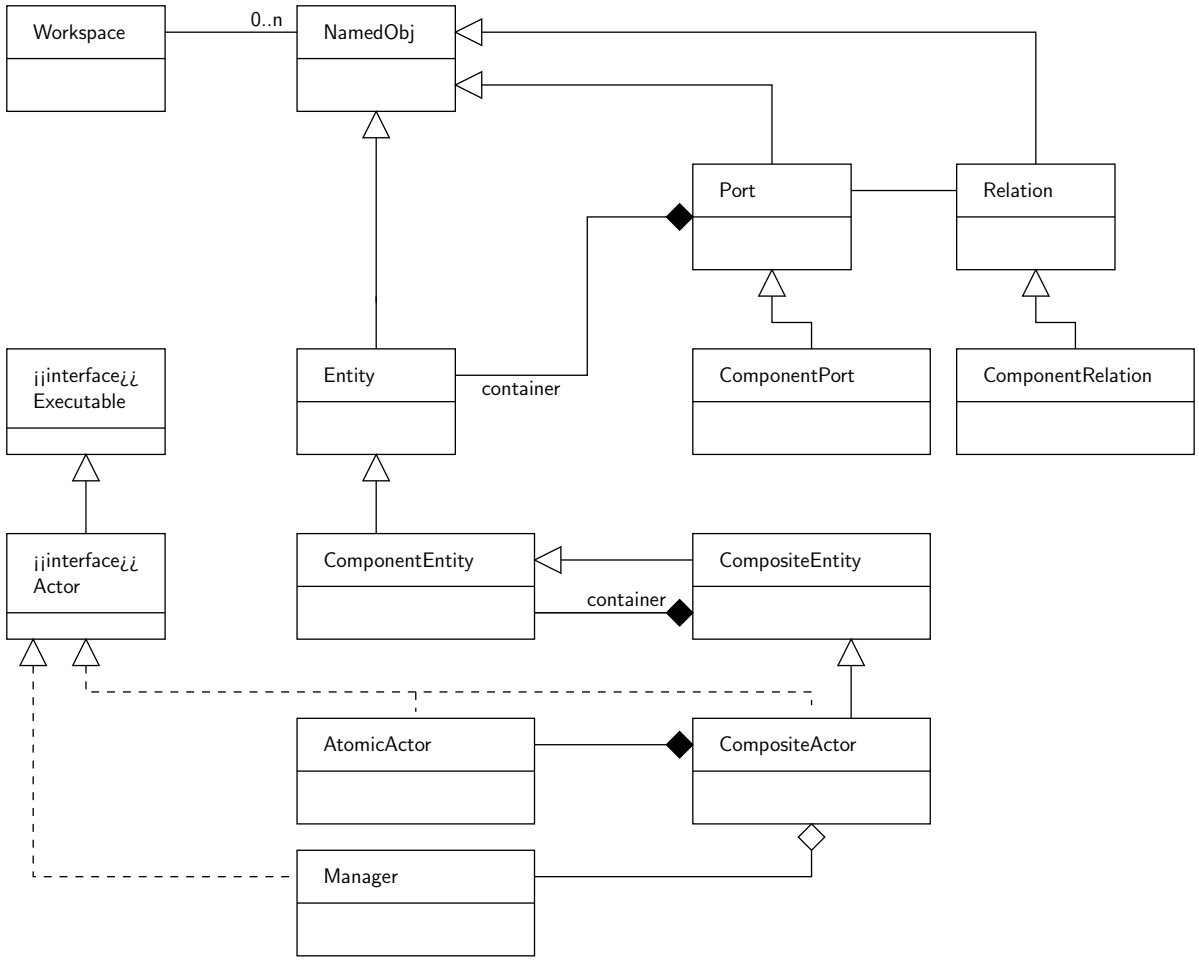
Application 5. Architecture for Interconnect System

Typical Ptolemy Application (see Brooks et al., 2008)



Application 5. Architecture for Interconnect System

Class diagram for modeling of system architectures in Ptolemy.



Application 5. Architecture for Interconnect System

From Individual Components to Networks of Components

Networks of components form graphs:

- **Graph.** A graph is an object that contains nodes and edges. Edges are accessed through the nodes that they connect.
- **Node.** A node is an object that is contained by a graph and is connected to other nodes by edges.
- **Edge.** An edge is an object that is contained by a graph and connects nodes.

  An edge has a "head" and a "tail" as if it was directed, but also has a method isDirected() that says whether or not the edge should be treated as directed.

Application 5. Architecture for Interconnect System

- **Port.** A Port is the interface of an Entity to any number of Relations. The role of a port is to aggregate a set of links to relations.

  Thus, for example, to represent a directed graph, entities can be created with two ports, one for incoming arcs and one for outgoing arcs.
- **Relation.** A Relation links ports, and therefore the entities that contain them.