

SoPra₂₀₁₂



**University of
Zurich^{UZH}**

Today

SoPra 2012

- Java RMI
- Java Swing
- Design Patterns
- JUnit Testing

Java RMI

SoPra 2012

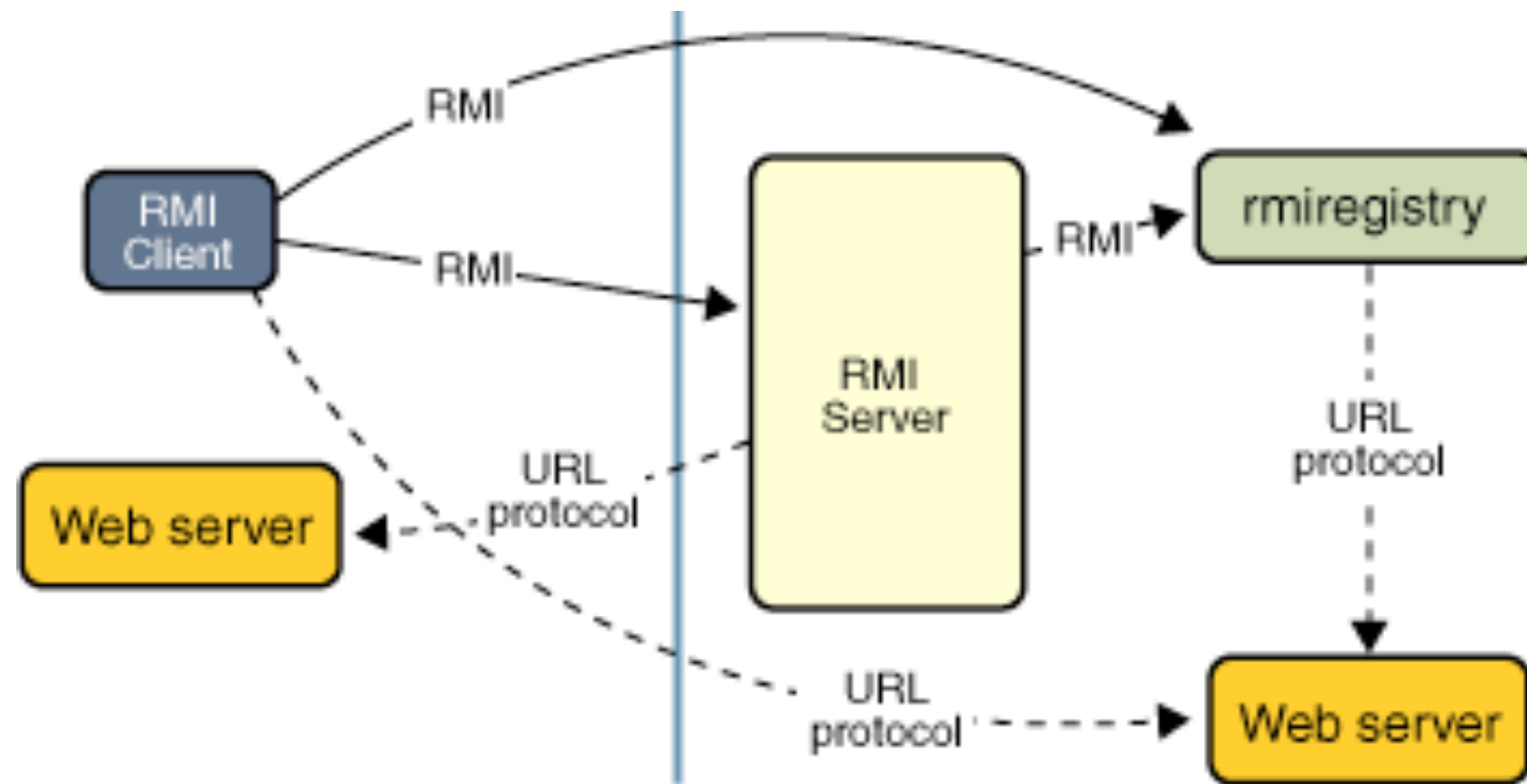
Java **R**emote **M**ethod **I**nvocation

The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine (JVM) to invoke methods on an object running in another JVM.

These JVMs may run on different computers!

Java RMI

SoPra 2012



Remote Objects Java RMI

...implement a **Remote Interface**:

- extends **java.rmi.Remote**
- throws **java.rmi.RemoteException**

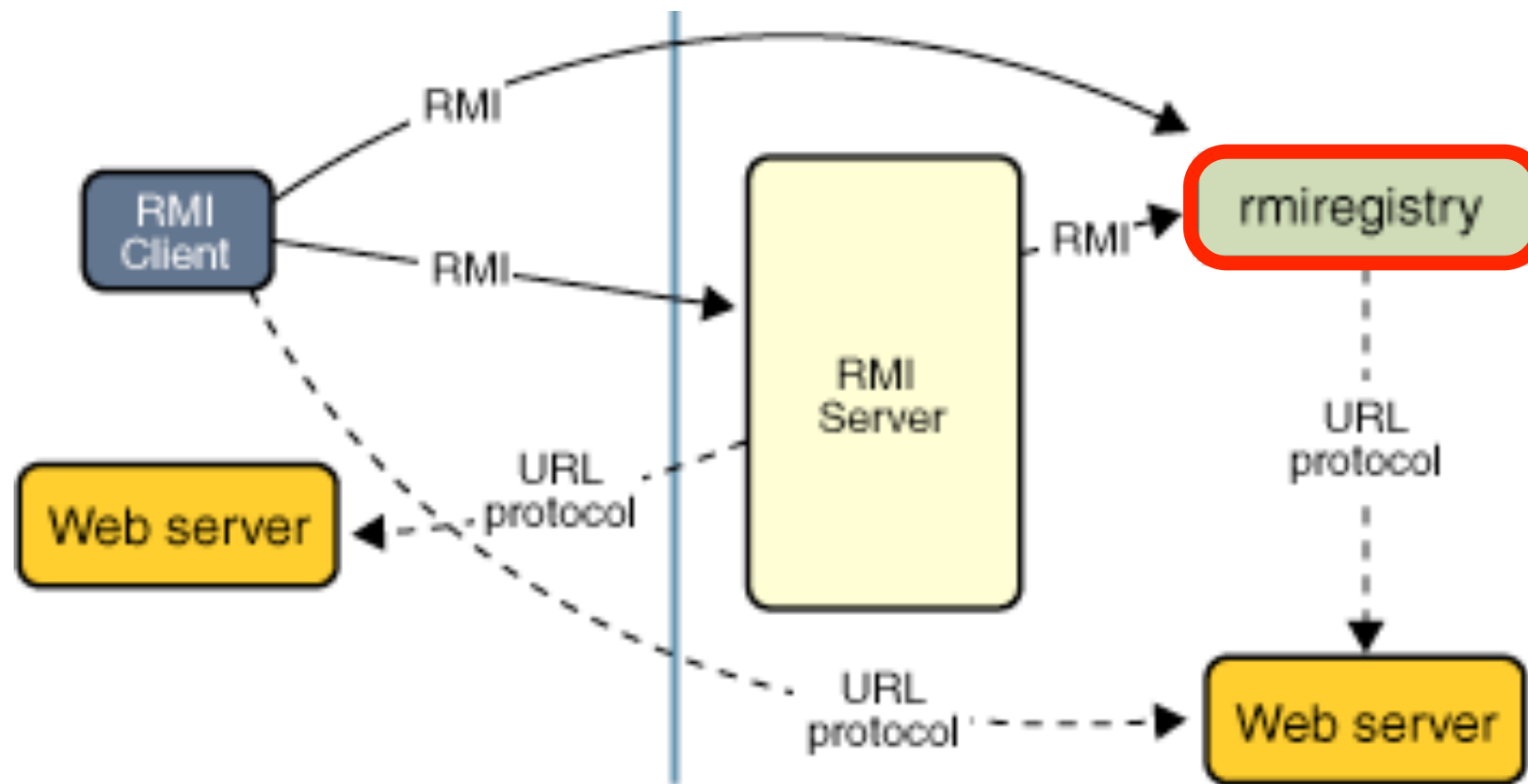
Example Java RMI

```
public interface Compute extends Remote {  
    public Task execute(Task task) throws RemoteException;  
}
```

```
public class ComputeEngine implements Compute {  
  
    public ComputeEngine() {  
        super();  
    }  
  
    public Task execute(Task task) {  
        return task.execute();  
    }  
}
```

RMI Registry

Java RMI



Example Java RMI

```
// Server
public static void main(String[] args) {
    try {
        String name = "Compute";
        Compute engine = new ComputeEngine();
        Compute proxy = (Compute)UnicastRemoteObject.exportObject(engine);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind(name, proxy);
        System.out.println("ComputeEngine bound");
    } catch (Exception e) {
        System.err.println("ComputeEngine exception:");
        e.printStackTrace();
    }
}
```


Example Java RMI

// Server

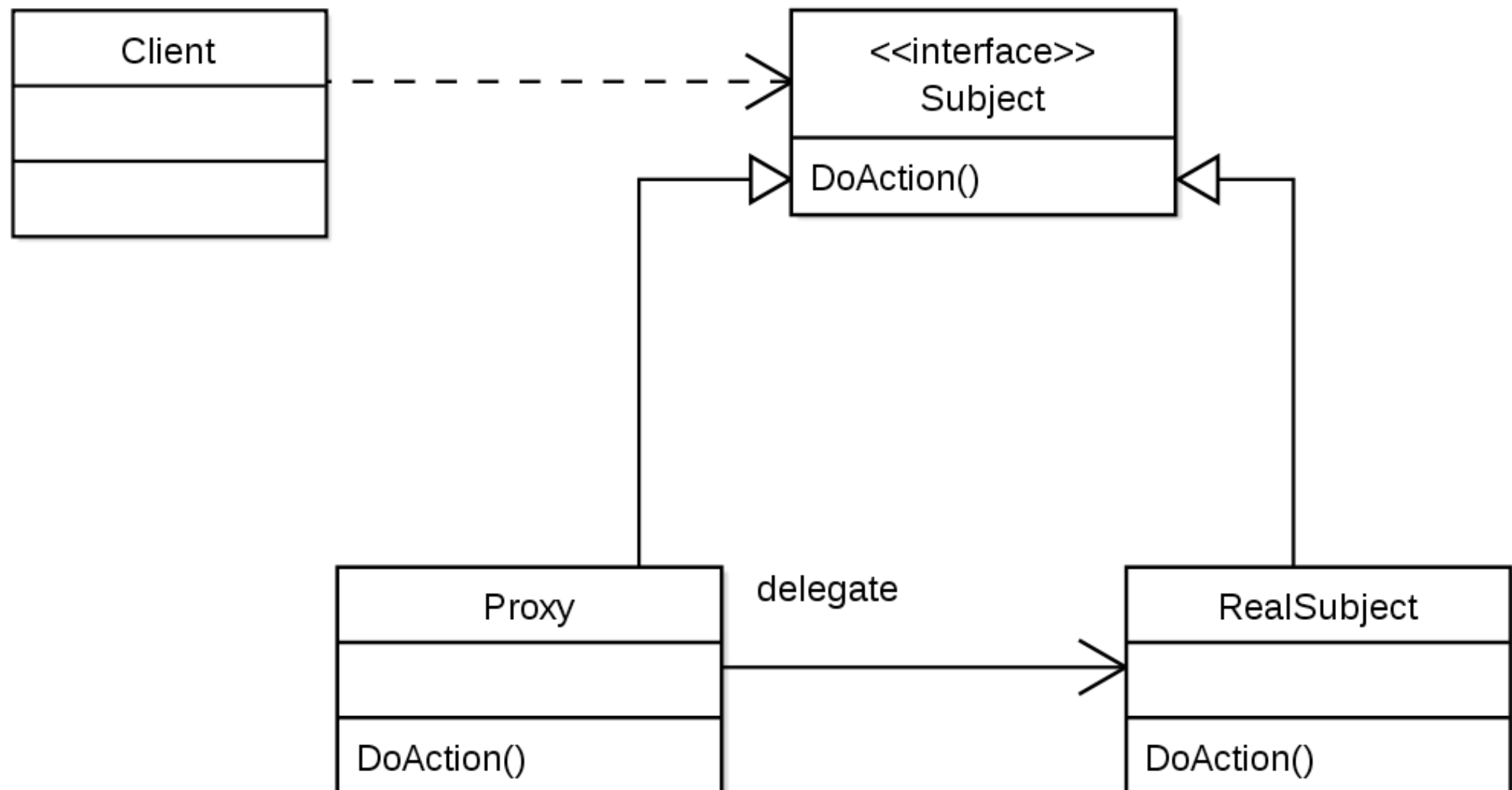
```
public static void main(String[] args) {
    try {
        String name = "Compute";
        Compute engine = new ComputeEngine();
        Compute proxy = (Compute)UnicastRemoteObject.exportObject(engine);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind(name, proxy);
        System.out.println("ComputeEngine bound");
    } catch (Exception e) {
        System.err.println("Compute exception:");
        e.printStackTrace();
    }
}
```

// Client

```
public static void main(String args[]) {
    try {
        String name = "Compute";
        Registry registry = LocateRegistry.getRegistry();
        Compute comp = (Compute)registry.lookup(name);
        /* ... */
    } catch (Exception e) {
        System.err.println("Compute exception:");
        e.printStackTrace();
    }
}
```

How does it work? Java RMI

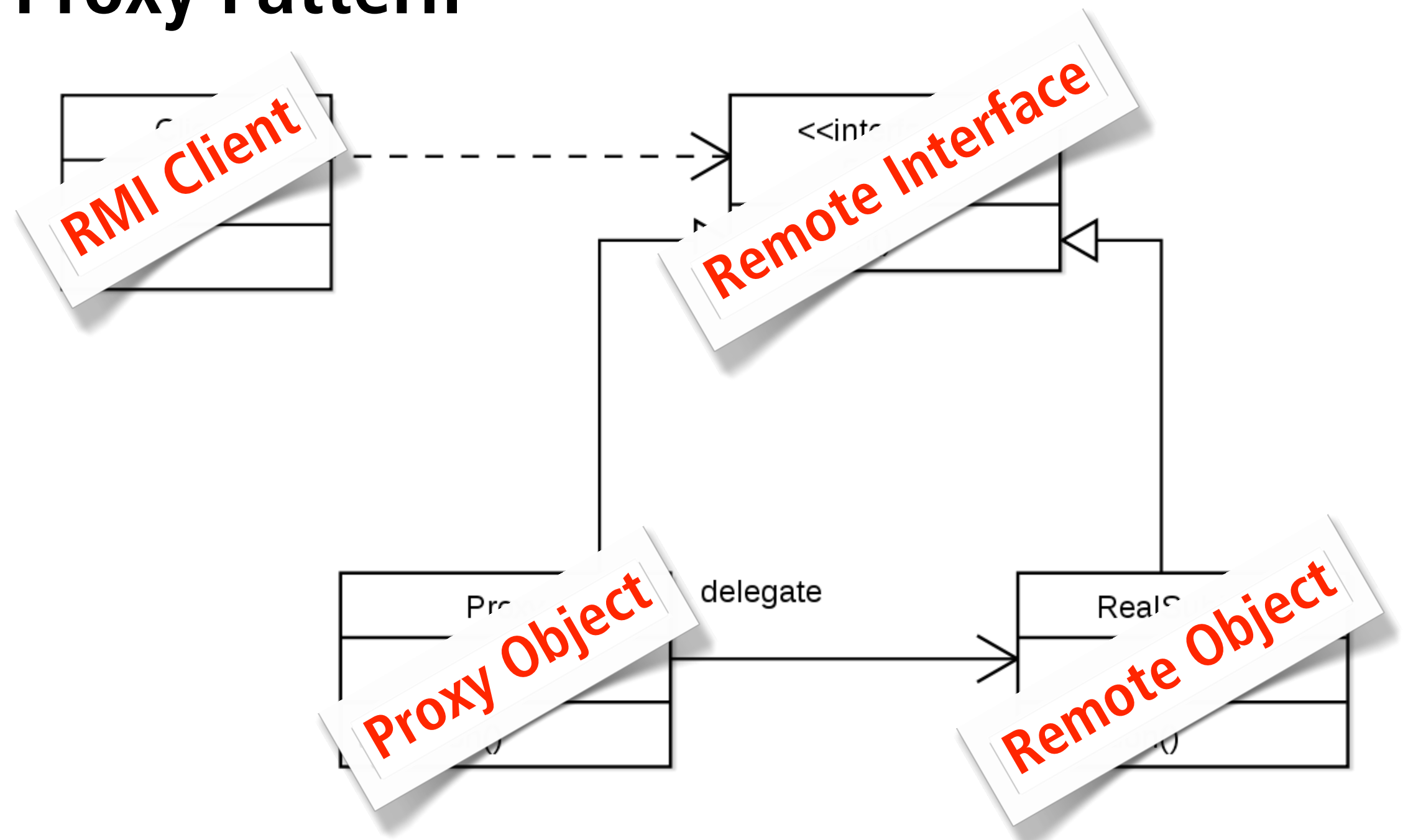
Proxy Pattern



How does it work?

Java RMI

Proxy Pattern



Java RMI

SoPra 2012

Java RMI Tutorial

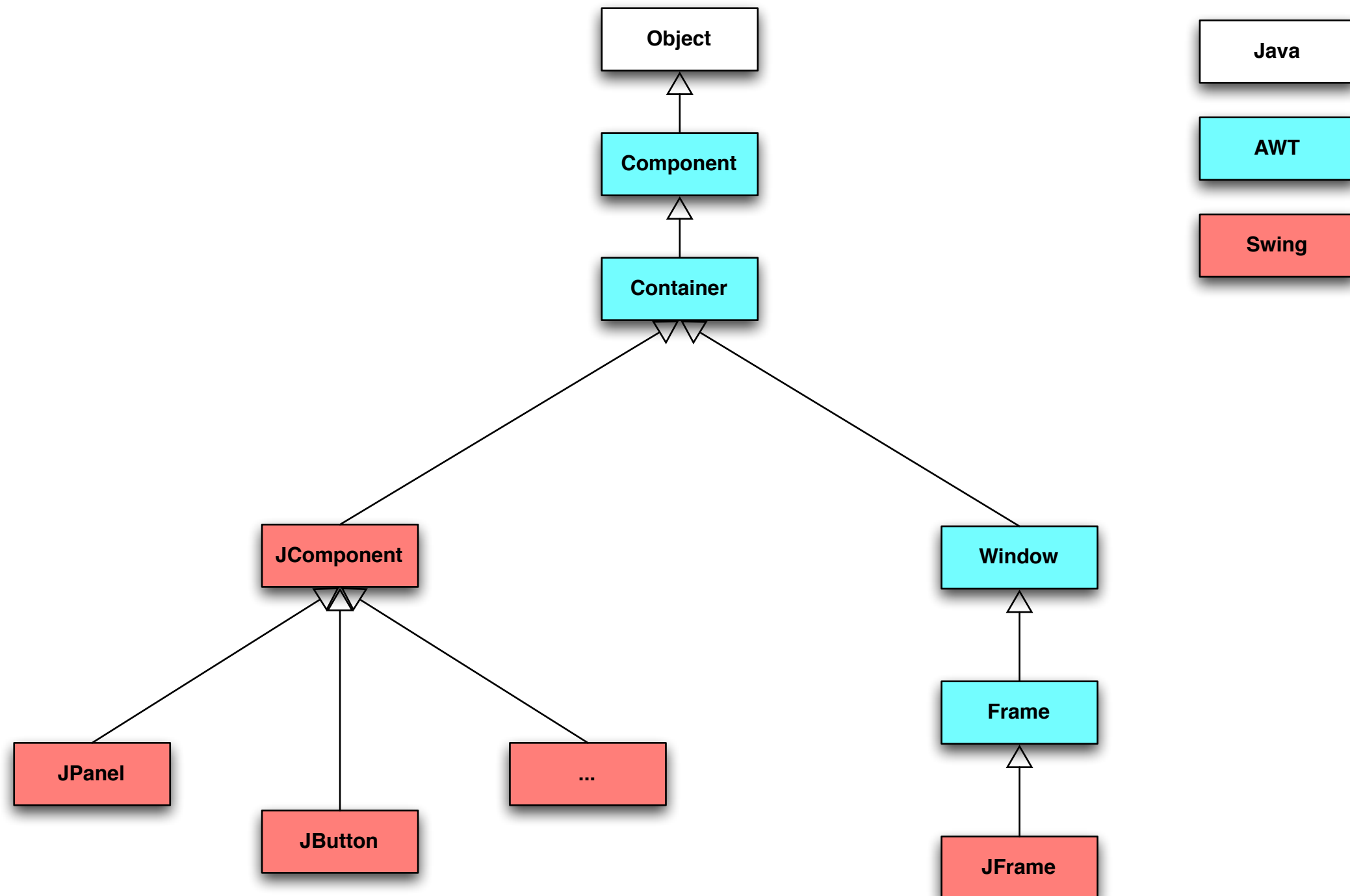
<http://seal.ifi.uzh.ch/sopra> ➡ Links

Java Swing SoPra 2012

- extension of the Abstract Window Toolkit (AWT)
- platform-independent GUI framework for Java
- set of GUI components
- extensible (e.g. SwingLabs SwingX)
- customizable
- configurable

Java Swing

SoPra 2012



JFrame

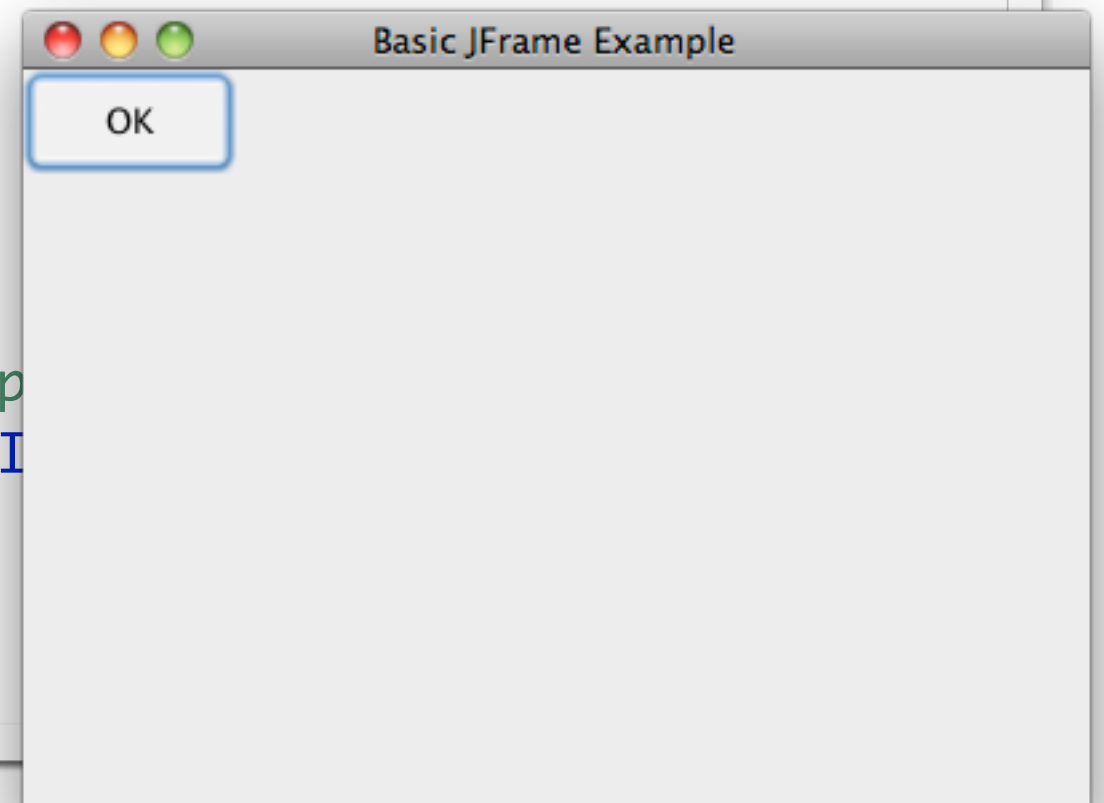
Java Swing

```
// create new JFrame object with title "Basic JFrame Example"
JFrame frame = new JFrame("Basic JFrame Example");
// create a new JButton object
JButton button = new JButton();
// text on JButton
button.setText("OK");
// set size of JButton
button.setSize(80, 40);
// disable layout manager
frame.setLayout(null);
// add JButton to JFrame
frame.add(button);
// size of JFrame
frame.setSize(400, 300);
// closing the JFrame closes the Java application
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// display JFrame
frame.setVisible(true);
```

JFrame

Java Swing

```
// create new JFrame object with title "Basic JFrame Example"
JFrame frame = new JFrame("Basic JFrame Example");
// create a new JButton object
JButton button = new JButton();
// text on JButton
button.setText("OK");
// set size of JButton
button.setSize(80, 40);
// disable layout manager
frame.setLayout(null);
// add JButton to JFrame
frame.add(button);
// size of JFrame
frame.setSize(400, 300);
// closing the JFrame closes the Java app
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// display JFrame
frame.setVisible(true);
```



Event Handling Java Swing

```
// add action listener as anonymous inner class
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 'OK' pressed!");
        /* ... */
    }
});
```

Event Handling Java Swing

```
// add action listener as anonymous inner class
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 'OK' pressed!");
        /* ... */
    }
});
```

```
// add action listener as 'normal' class...
button.addActionListener(new OkButtonActionListener());
```

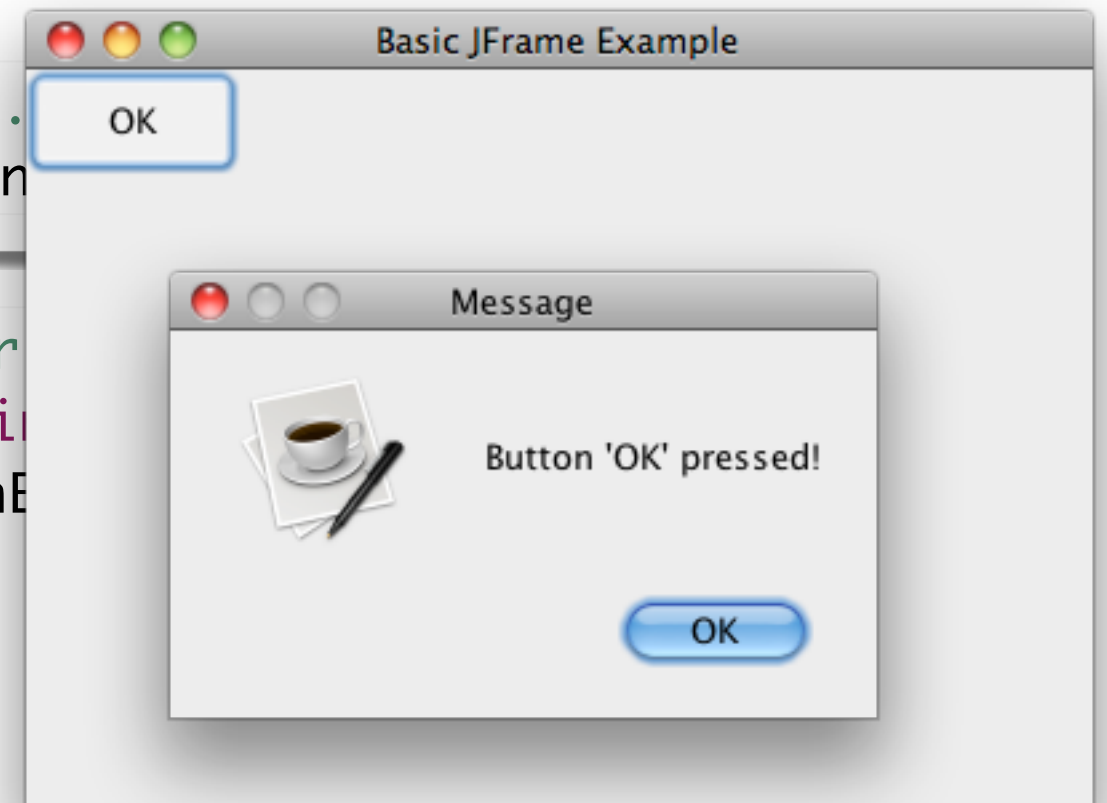
```
// ...implementing the ActionListener interface
public class OkButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 'OK' pressed!");
        /* ... */
    }
}
```

Event Handling Java Swing

```
// add action listener as anonymous inner class
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 'OK' pressed!");
        /* ... */
    }
});
```

```
// add action listener as 'normal' class...
button.addActionListener(new OkButtonAction
```

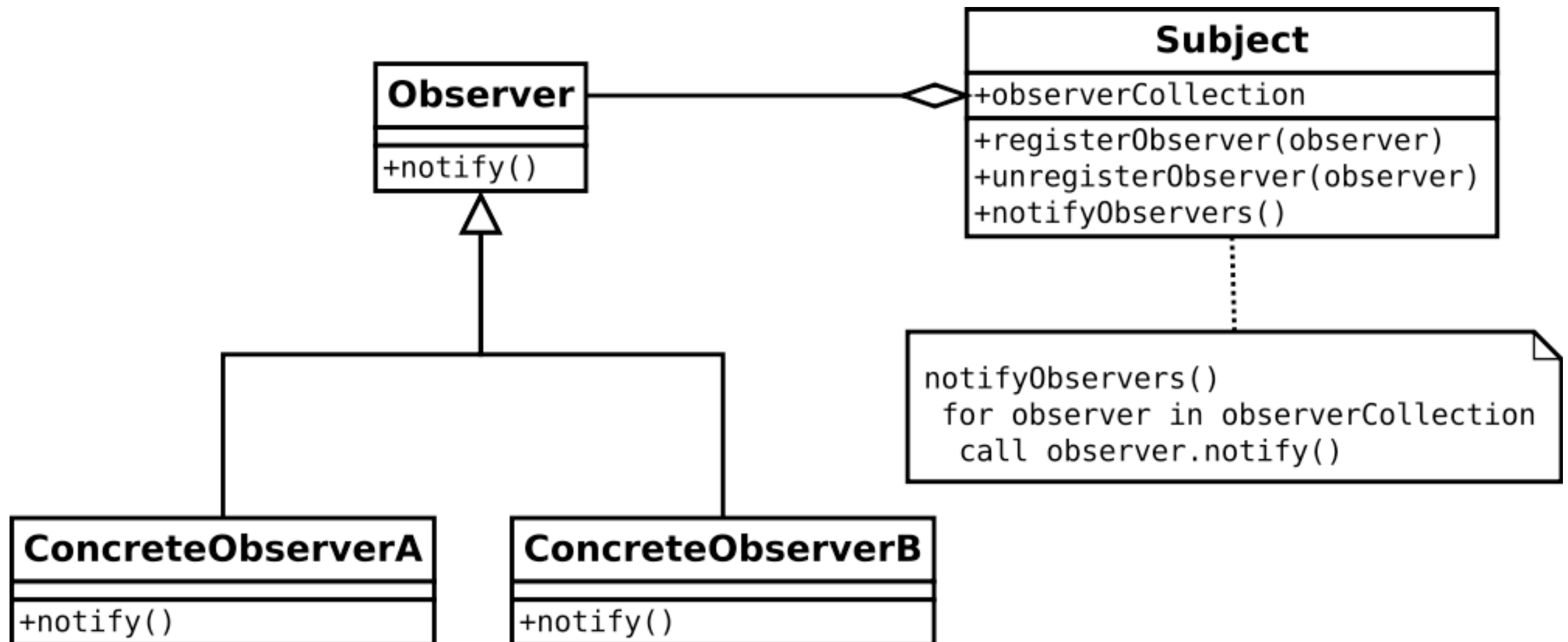
```
// ...implementing the ActionListener
public class OkButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button 'OK' pressed!");
        /* ... */
    }
}
```



How does it work?

Event Handling

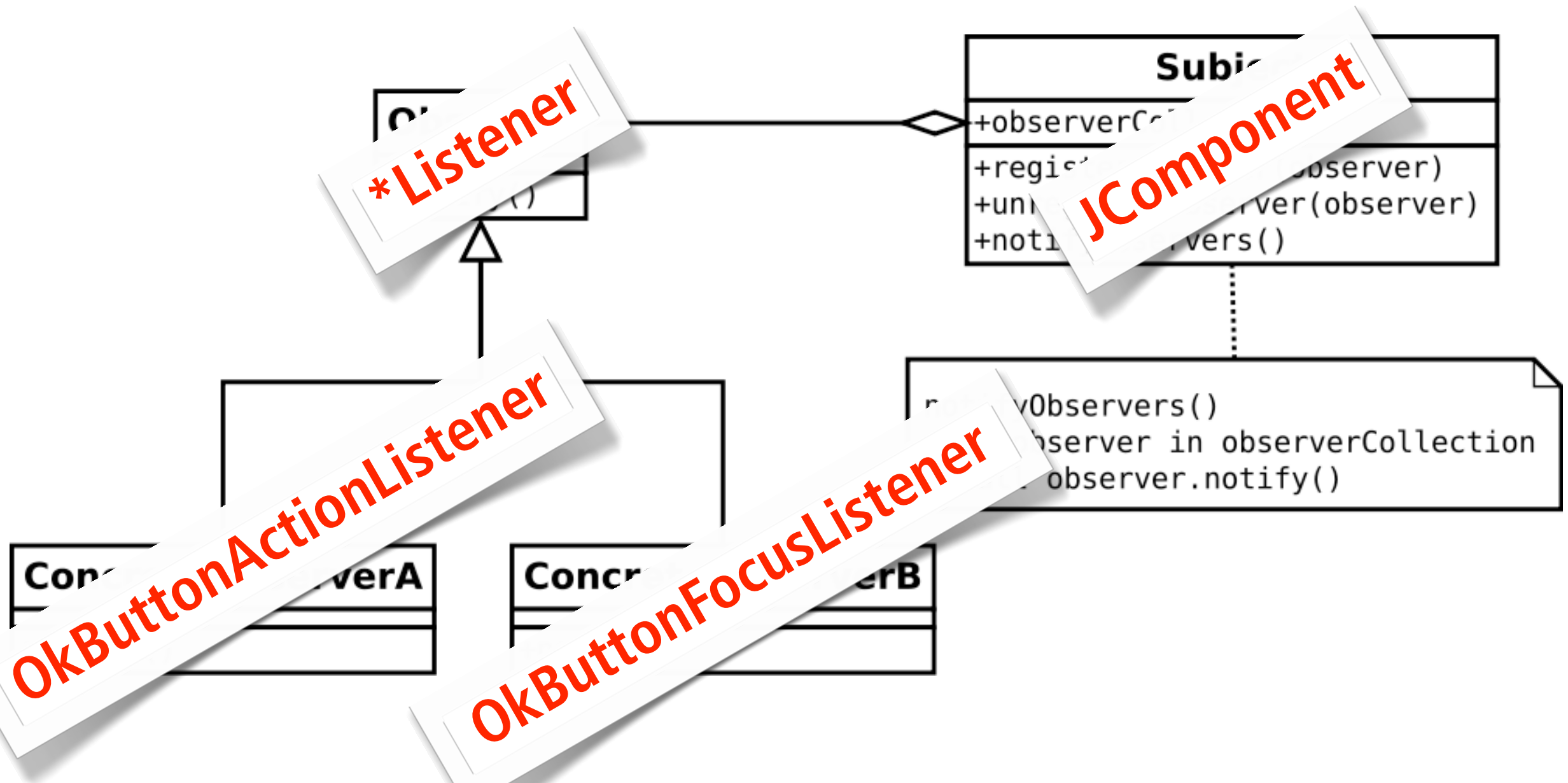
Observer Pattern



How does it work?

Event Handling

Observer Pattern



SwingLabs SwingX_{Java Swing}

Example I: JPanel + ImagePainter

```
// ...  
JFrame frame = new JFrame("ImagePanelDemo");  
  
JPanel imagePanel = new JPanel();  
ImagePainter painter = new ImagePainter(ImageIO.read(imageUrl));  
imagePanel.setBackgroundPainter(painter);  
  
frame.add(imagePanel);  
// ...
```

SwingLabs SwingX^{Java Swing}

Example I: JPanel + ImagePainter

```
// ...  
JFrame frame = new JFrame("ImagePanelDemo");  
  
JPanel imagePanel = new JPanel();  
ImagePainter painter = new ImagePainter();  
imagePanel.setBackgroundPainter(painter);  
  
frame.add(imagePanel);  
// ...
```



SwingLabs SwingX_{Java Swing}

Example II: JXBusyLabel

```
JFrame frame = new JFrame("BusyLabelDemo");
```

```
JXBusyLabel label = new JXBusyLabel();
```

```
frame.add(label);
```

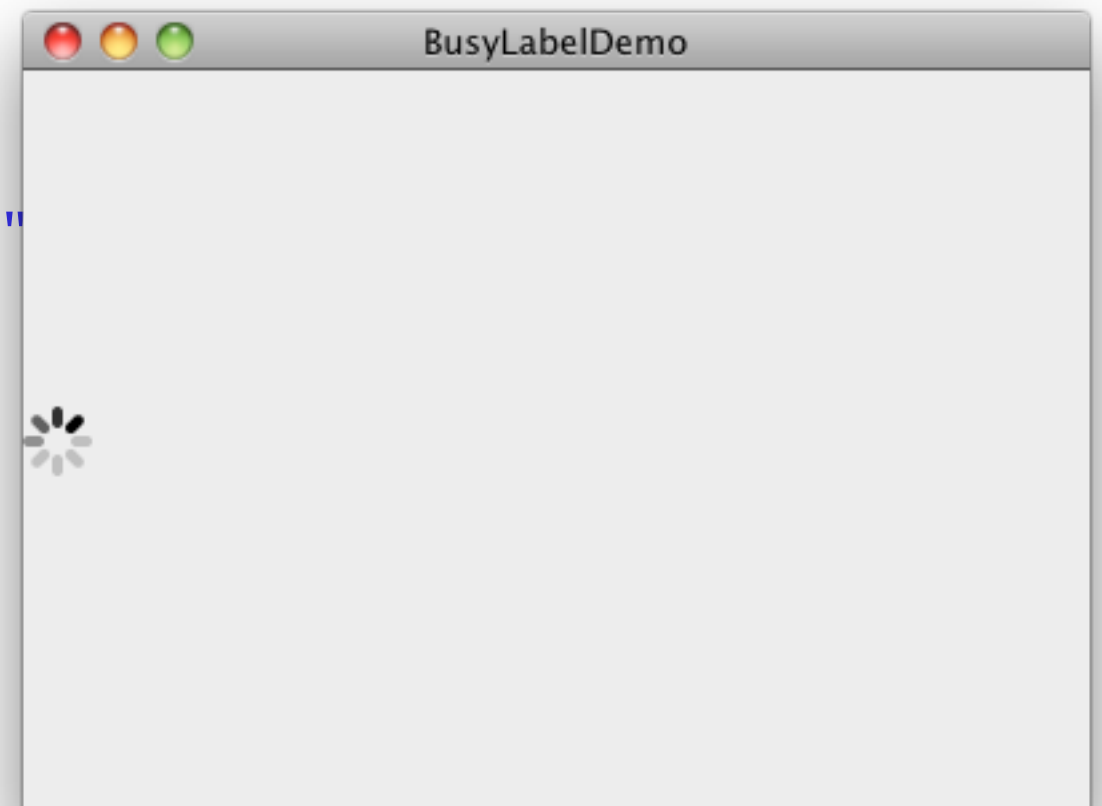
```
label.setBusy(true);
```

```
// ...
```


SwingLabs SwingX_{Java Swing}

Example II: JXBusyLabel

```
JFrame frame = new JFrame("BusyLabelDemo")  
  
JXBusyLabel label = new JXBusyLabel();  
  
frame.add(label);  
label.setBusy(true);  
// ...
```



Java Swing SoPra 2012

- **Java Swing Tutorial**
- **SwingLabs SwingX**

<http://seal.ifi.uzh.ch/sopra> ➡ Links

Design Patterns SoPra 2012

A **design pattern** is a general reusable solution to a commonly occurring problem in software design.

A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Design Patterns SoPra 2012

- **State Pattern**
- Observer Pattern
- Singleton Pattern
- Command Pattern
- Decorater Pattern
- Composite Pattern
- Proxy Pattern
- ...

State Pattern

Design Patterns

- Etliche Zustände eines Objektes
- Zustandsübergänge verbunden mit komplizierten Bedingungen
- Spielzustände !
- Spielregeln bestimmen Zustandsübergänge
- Naiver Ansatz: Grosses Switch-Case Statement, tief verschachteltes if-else-if-else-if-... Konstrukt
- Beispiele: Parser und Scanner Klassen aus Eclipse

State Pattern

Design Patterns

```
5109 case 30 : if (DEBUG) { System.out.println("Type ::= PrimitiveType"); } //$NON-NLS-1$
5110     consumePrimitiveType();
5111     break;
5112
5113 case 44 : if (DEBUG) { System.out.println("ReferenceType ::= ClassOrInterfaceType"); } //$NON-NLS-1$
5114     consumeReferenceType();
5115     break;
5116
5117 case 48 : if (DEBUG) { System.out.println("ClassOrInterface ::= Name"); } //$NON-NLS-1$
5118     consumeClassOrInterfaceName();
5119     break;
5120
5121 case 49 : if (DEBUG) { System.out.println("ClassOrInterface ::= GenericType DOT Name"); } //$NON-NLS-1$
5122     consumeClassOrInterface();
5123     break;
5124
5125 case 50 : if (DEBUG) { System.out.println("GenericType ::= ClassOrInterface TypeArguments"); } //$NON-NLS-1$
5126     consumeGenericType();
5127     break;
5128
5129 case 51 : if (DEBUG) { System.out.println("ArrayTypeWithTypeArgumentsName ::= GenericType DOT Name"); } //$NON-NLS-1$
5130     consumeArrayTypeWithTypeArgumentsName();
5131     break;
5132
5133 case 52 : if (DEBUG) { System.out.println("ArrayType ::= PrimitiveType Dims"); } //$NON-NLS-1$
5134     consumePrimitiveArrayType();
5135     break;
5136
5137 case 53 : if (DEBUG) { System.out.println("ArrayType ::= Name Dims"); } //$NON-NLS-1$
5138     consumeNameArrayType();
5139     break;
5140
5141 case 54 : if (DEBUG) { System.out.println("ArrayType ::= ArrayTypeWithTypeArgumentsName Dims"); } //$NON-NLS-1$
5142     consumeGenericTypeFromArrayType();
5143     break;
5144
```

State Pattern

Design Patterns

```
6105         break;
6106
6107     case 456 : if (DEBUG) { System.out.println("AssignmentOperator ::= UNSIGNED_RIGHT_SHIFT_EQUAL"); } //$NON-NLS-1$
6108         consumeAssignmentOperator(UNSIGNED_RIGHT_SHIFT);
6109         break;
6110
6111     case 457 : if (DEBUG) { System.out.println("AssignmentOperator ::= AND_EQUAL"); } //$NON-NLS-1$
6112         consumeAssignmentOperator(AND);
6113         break;
6114
6115     case 458 : if (DEBUG) { System.out.println("AssignmentOperator ::= XOR_EQUAL"); } //$NON-NLS-1$
6116         consumeAssignmentOperator(XOR);
6117         break;
6118
6119     case 459 : if (DEBUG) { System.out.println("AssignmentOperator ::= OR_EQUAL"); } //$NON-NLS-1$
6120         consumeAssignmentOperator(OR);
6121         break;
6122
6123     case 463 : if (DEBUG) { System.out.println("Expressionopt ::="); } //$NON-NLS-1$
6124         consumeEmptyExpression();
6125         break;
6126
6127     case 468 : if (DEBUG) { System.out.println("ClassBodyDeclarationsopt ::="); } //$NON-NLS-1$
6128         consumeEmptyClassBodyDeclarationsopt();
6129         break;
6130
6131     case 469 : if (DEBUG) { System.out.println("ClassBodyDeclarationsopt ::= NestedType..."); } //$NON-NLS-1$
6132         consumeClassBodyDeclarationsopt();
6133         break;
6134
6135     case 470 : if (DEBUG) { System.out.println("Modifiersopt ::="); } //$NON-NLS-1$
6136         consumeDefaultModifiers();
6137         break;
6138
6139     case 471 : if (DEBUG) { System.out.println("Modifiersopt ::= Modifiers"); } //$NON-NLS-1$
6140         consumeModifiers();
6141         break;
6142
6143     case 472 : if (DEBUG) { System.out.println("BlockStatementsopt ::="); } //$NON-NLS-1$
6144         consumeEmptyBlockStatementsopt();
6145         break;
6146
6147     case 474 : if (DEBUG) { System.out.println("Dimsopt ::="); } //$NON-NLS-1$
6148         consumeEmptyDimsopt();
6149         break;
```


State Pattern

Design Patterns

```
6750
6751 case 694 : if (DEBUG) { System.out.println("MarkerAnnotation ::= AnnotationName"); } //$NON-NLS-1$
6752     consumeMarkerAnnotation() ;
6753     break;
6754
6755 case 695 : if (DEBUG) { System.out.println("SingleMemberAnnotationMemberValue ::= MemberValue"); } //$NON-NLS-1$
6756     consumeSingleMemberAnnotationMemberValue() ;
6757     break;
6758
6759 case 696 : if (DEBUG) { System.out.println("SingleMemberAnnotation ::= AnnotationName LPAREN..."); } //$NON-NLS-1$
6760     consumeSingleMemberAnnotation() ;
6761     break;
6762
6763 case 697 : if (DEBUG) { System.out.println("RecoveryMethodHeaderName ::= Modifiersopt TypeParameters"); } //$NON-NLS-1$
6764     consumeRecoveryMethodHeaderNameWithTypeParameters();
6765     break;
6766
6767 case 698 : if (DEBUG) { System.out.println("RecoveryMethodHeaderName ::= Modifiersopt Type..."); } //$NON-NLS-1$
6768     consumeRecoveryMethodHeaderName();
6769     break;
6770
6771 case 699 : if (DEBUG) { System.out.println("RecoveryMethodHeader ::= RecoveryMethodHeaderName..."); } //$NON-NLS-1$
6772     consumeMethodHeader();
6773     break;
6774
6775 case 700 : if (DEBUG) { System.out.println("RecoveryMethodHeader ::= RecoveryMethodHeaderName..."); } //$NON-NLS-1$
6776     consumeMethodHeader();
6777     break;
6778
```


State Pattern

Design Patterns

```
case 'e' : //else extends
switch (length) {
case 4 :
    if ((data[++index] == 'l') && (data[++index] == 's') && (data[++index] == 'e'))
        return TokenNameelse;
    else if ((data[index] == 'n')
        && (data[++index] == 'u')
        && (data[++index] == 'm')) {
        if (this.sourceLevel >= ClassFileConstants.JDK1_5) {
            return TokenNameenum;
        } else {
            this.useEnumAsAnIdentifier = true;
            return TokenNameidentifier;
        }
    } else {
        return TokenNameidentifier;
    }
case 7 :
    if ((data[++index] == 'x')
        && (data[++index] == 't')
        && (data[++index] == 'e')
        && (data[++index] == 'n')
        && (data[++index] == 'd')
        && (data[++index] == 's'))
        return TokenNameextends;
    else
        return TokenNameidentifier;
default :
    return TokenNameidentifier;
}

case 'f' : //final finally float for false
switch (length) {
case 3 :
    if ((data[++index] == 'o') && (data[++index] == 'r'))
        return TokenNamefor;
    else
        return TokenNameidentifier;
}
```

State Pattern

Design Patterns

```
,  
if ((c1 = ScannerHelper.getNumericValue(this.source[this.currentPosition++])) > 15  
    || c1 < 0  
    || (c2 = ScannerHelper.getNumericValue(this.source[this.currentPosition++])) > 15  
    || c2 < 0  
    || (c3 = ScannerHelper.getNumericValue(this.source[this.currentPosition++])) > 15  
    || c3 < 0  
    || (c4 = ScannerHelper.getNumericValue(this.source[this.currentPosition++])) > 15  
    || c4 < 0){  
    throw new InvalidInputException(INVALID_UNICODE_ESCAPE);  
}
```

State Pattern

Design Patterns

```
if (getNextChar('d', 'D') >= 0) {
    return TokenNameDoubleLiteral;
} else { //make the distinction between octal and float ....
    boolean isInteger = true;
    if (getNextChar('.')) {
        isInteger = false;
        while (getNextCharAsDigit()){/*empty*/}
    }
    if (getNextChar('e', 'E') >= 0) { // consume next character
        isInteger = false;
        this.unicodeAsBackSlash = false;
        if (((this.currentCharacter = this.source[this.currentPosition++]) == '\\')
            && (this.source[this.currentPosition] == 'u')) {
            getNextUnicodeChar();
        } else {
            if (this.withoutUnicodePtr != 0) {
                unicodeStore();
            }
        }
    }

    if ((this.currentCharacter == '-')
        || (this.currentCharacter == '+')) { // consume next character
        this.unicodeAsBackSlash = false;
        if (((this.currentCharacter = this.source[this.currentPosition++]) == '\\')
            && (this.source[this.currentPosition] == 'u')) {
            getNextUnicodeChar();
        } else {
            if (this.withoutUnicodePtr != 0) {
                unicodeStore();
            }
        }
    }
}
if (!ScannerHelper.isDigit(this.currentCharacter))
    throw new InvalidInputException(INVALID_FLOAT);
while (getNextCharAsDigit()){/*empty*/}
}
if (getNextChar('f', 'F') >= 0)
    return TokenNameFloatingPointLiteral;
if (getNextChar('d', 'D') >= 0 || !isInteger)
    return TokenNameDoubleLiteral;
return TokenNameIntegerLiteral;
```

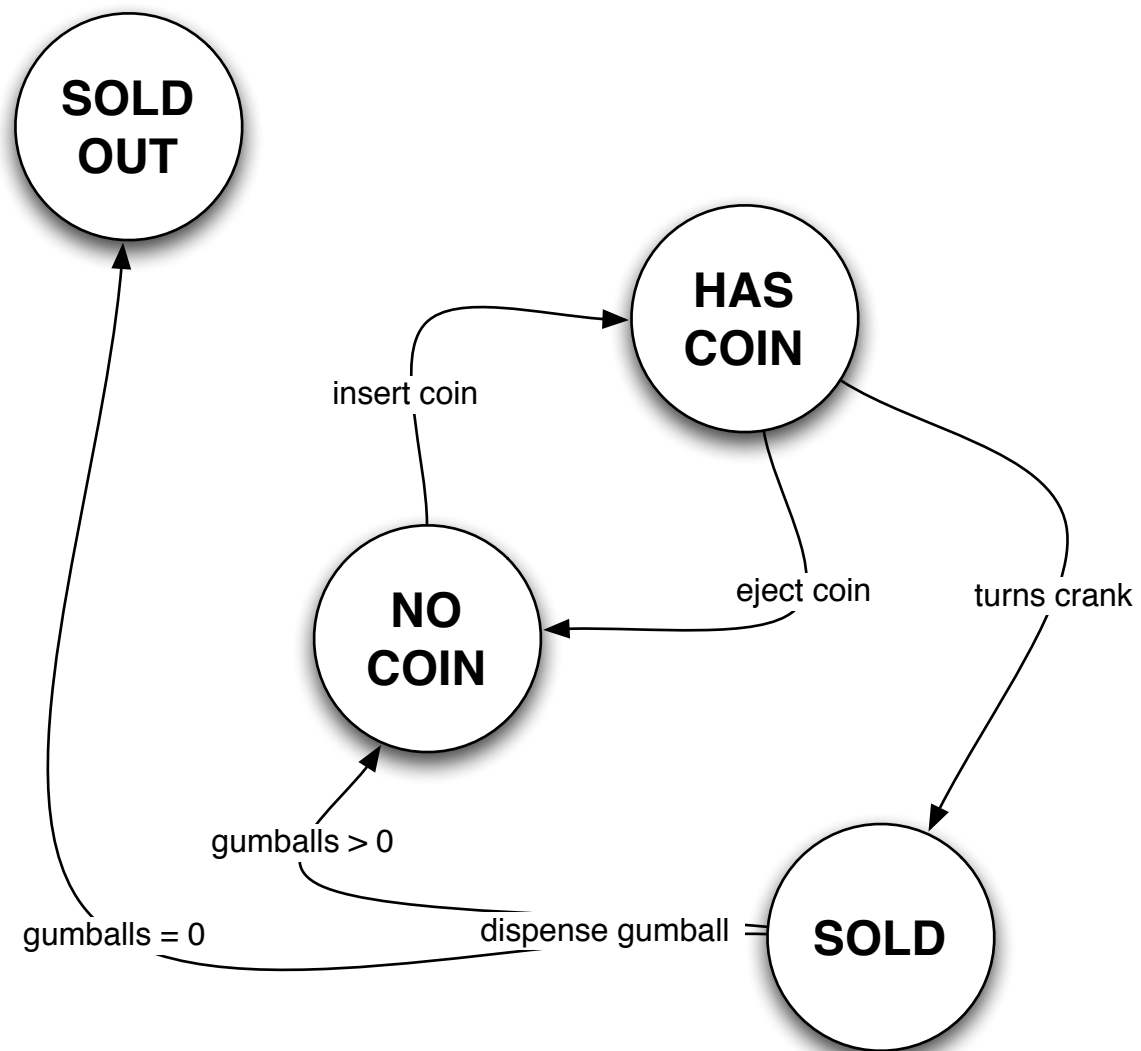
State Pattern

Design Patterns

- Zustand und damit verbundenes Verhalten wird in eigenen Klassen modelliert
- Für jeden Zustand eine Klasse, die Verhalten in diesem Zustand definiert
- Regeln werden ebenfalls in diesen Zustandsklassen gekapselt
- → Ein Zustand kennt seinen Nachfolger

Example

State Pattern



Example

State Pattern

```
public class GumballMachine {  
  
    final static int SOLD_OUT = 0;  
    final static int NO_COIN = 1;  
    final static int HAS_COIN = 2;  
    final static int SOLD = 3;  
  
    int state = SOLD_OUT;  
    int count = 0;  
  
    public GumballMachine(int count) {  
        this.count = count;  
        if (count > 0) {  
            state = NO_COIN;  
        }  
    }  
}
```

Example

State Pattern

```
public class GumballMachine {
```

```
    final static int SOLD_OUT = 0;  
    final static int NO_COIN = 1;  
    final static int HAS_COIN = 2;  
    final static int SOLD = 3;
```

```
    int state;  
    int count;
```

```
    public GumballMachine()  
    {  
        this.state = NO_COIN;  
        if (count == 0)  
            state = SOLD_OUT;  
    }  
}
```

```
    public void insertCoin() {  
        if (state == HAS_COIN) {  
            System.out.println("You can't insert another coin");  
        } else if (state == NO_COIN) {  
            state = HAS_COIN;  
            System.out.println("You inserted a coin");  
        } else if (state == SOLD_OUT) {  
            System.out.println("You can't insert a coin, the machine is sold out");  
        } else if (state == SOLD) {  
            System.out.println("Please wait, we're already giving you a gumball");  
        }  
    }  
}
```

Example

State Pattern

```
public void ejectCoin() {  
    if (state == HAS_COIN) {  
        System.out.println("Coin returned");  
        state = NO_COIN;  
    } else if (state == NO_COIN) {  
        System.out.println("You haven't inserted a coin");  
    } else if (state == SOLD) {  
        System.out.println("Sorry, you already turned the crank");  
    } else if (state == SOLD_OUT) {  
        System.out.println("You can't eject, you haven't inserted a coin yet");  
    }  
}
```


Example

State Pattern

```
public void ejectCoin() {
    if (state == HAS_COIN) {
        System.out.println("Coin returned");
    }
}

public void dispense() {
    if (state == SOLD) {
        System.out.println("A gumball comes rolling out the slot");
        count = count - 1;
        if (count == 0) {
            System.out.println("Oops, out of gumballs!");
            state = SOLD_OUT;
        } else {
            state = NO_COIN;
        }
    } else if (state == NO_COIN) {
        System.out.println("You need to pay first");
    } else if (state == SOLD_OUT) {
        System.out.println("No gumball dispensed");
    } else if (state == HAS_COIN) {
        System.out.println("No gumball dispensed");
    }
}
```

Example

State Pattern

Change Request:

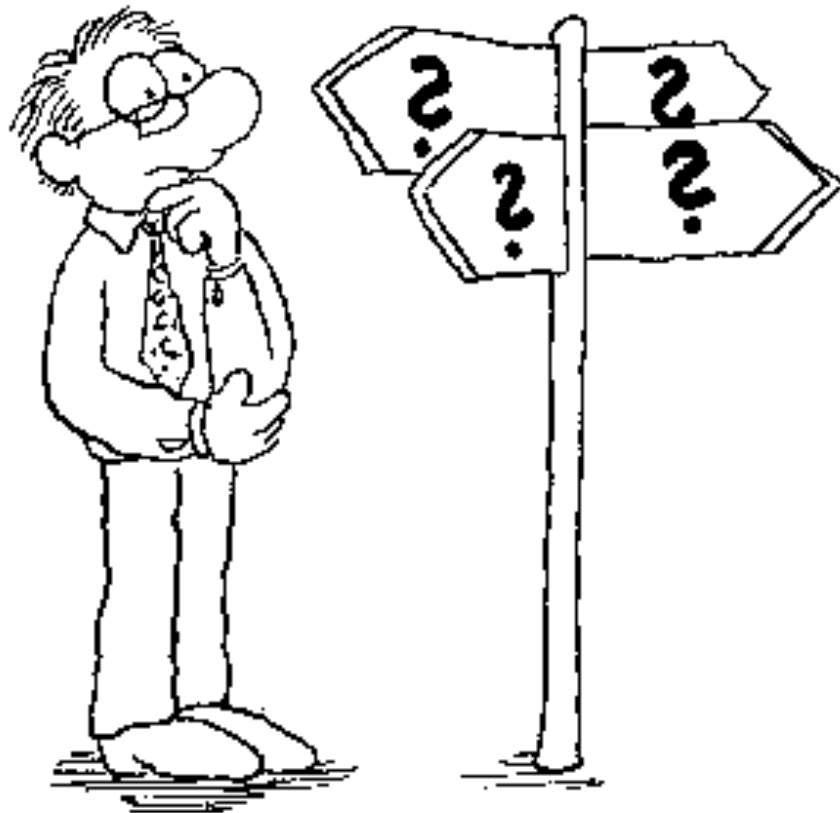


One in Ten get a FREE GUMBALL!



Example

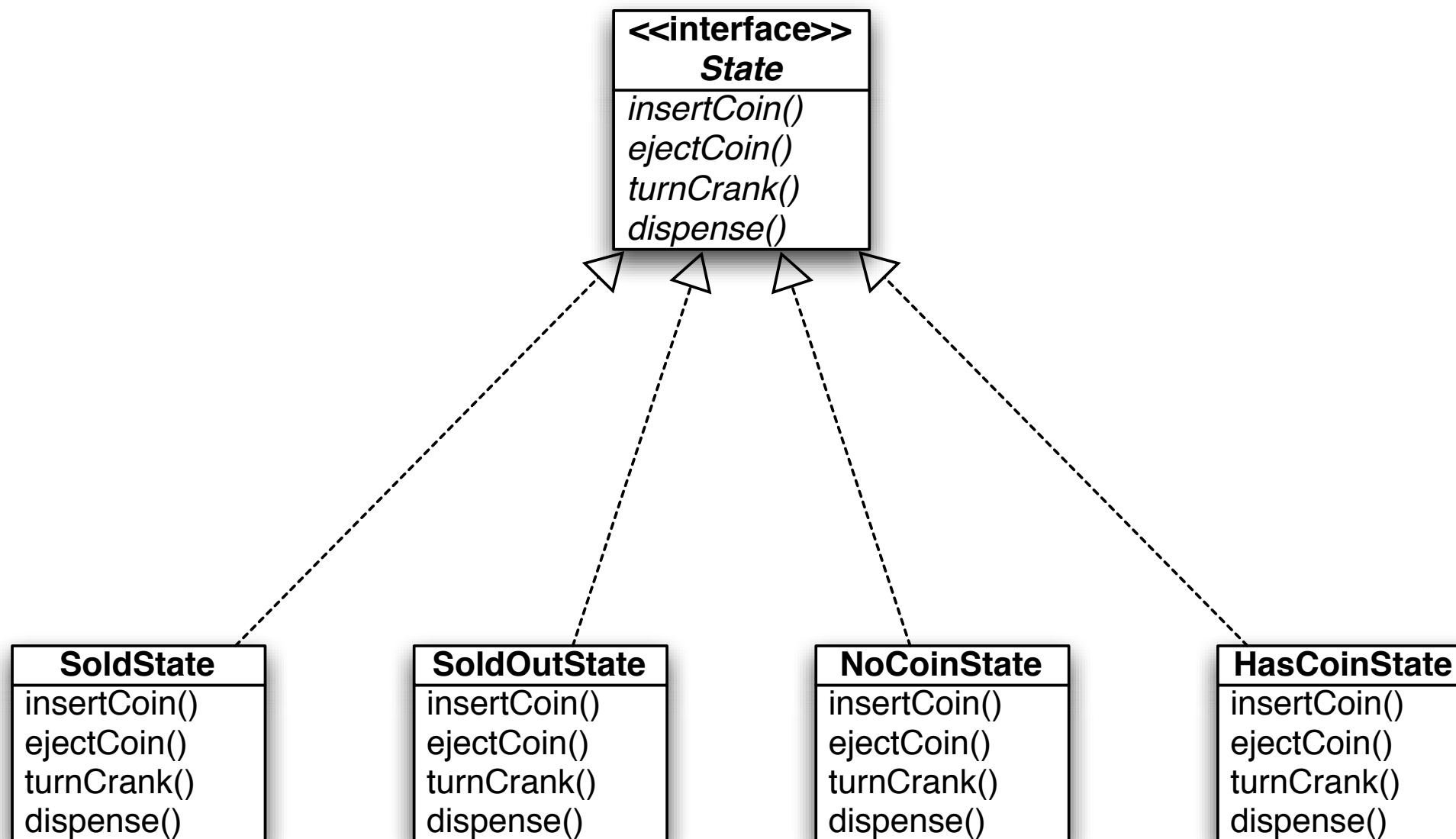
State Pattern



```
public class GumballMachine {  
  
    final static int SOLD_OUT = 0;  
    final static int NO_QUARTER = 1;  
    final static int HAS_QUARTER = 2;  
    final static int SOLD = 3;  
    // CHANGE NEEDED  
  
    int state = SOLD_OUT;  
    int count = 0;  
  
    public GumballMachine(int count) {  
    }  
  
    public void insertQuarter() {  
        // CHANGE NEEDED  
    }  
  
    public void ejectQuarter() {  
        // CHANGE NEEDED  
    }  
  
    public void turnCrank() {  
        // CHANGE NEEDED  
    }  
  
    public void dispense() {  
        // CHANGE NEEDED  
    }  
}
```

Example State Pattern

New Design: **State Pattern**



Example

State Pattern

New Design

SoldState
insertCoin() ejectCoin() turnCrank() dispense()

```
public class NoCoinState implements State {
    GumballMachine gumballMachine;

    public NoCoinState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

    public void insertCoin() {
        System.out.println("You inserted a quarter");
        gumballMachine.setState(gumballMachine.getHasCoinState());
    }

    public void ejectCoin() {
        System.out.println("You haven't inserted a coin");
    }

    public void turnCrank() {
        System.out.println("You turned, but there's no coin");
    }

    public void dispense() {
        System.out.println("You need to pay first");
    }
}
```

Example

State Pattern

```
public class GumballMachine {  
  
    State soldOutState;  
    State noCoinState;  
    State hasCoinState;  
    State soldState;  
  
    State state = soldOutState;  
    int count = 0;  
  
    public GumballMachine(int numberGumballs) {  
        soldOutState = new SoldOutState(this);  
        noCoinState = new NoQuarterState(this);  
        hasCoinState = new HasCoinState(this);  
        soldState = new SoldState(this);  
  
        this.count = numberGumballs;  
        if (numberGumballs > 0) {  
            state = noCoinState;  
        }  
    }  
}
```

Example

State Pattern

```
public class GumballMachine {

    State soldOutState;
    State noCoinState;
    State hasCoinState;
    State soldState;

    State state = soldOutState;
    int count = 0;

    public GumballMachine(int numberGumballs) {
        soldOutState = new SoldOutState(this);
        noCoinState = new NoQuarterState(this);
        hasCoinState = new HasCoinState(this);
        soldState = new SoldState(this);

        this.count = numberGumballs;
        if (numberGumballs > 0) {
            state = noCoinState;
        }
    }
}
```

```
public void insertCoin() {
    state.insertCoin();
}

public void ejectCoin() {
    state.ejectCoin();
}

public void turnCrank() {
    state.turnCrank();
    state.dispense();
}
```

Example

State Pattern

```
public class GumballMachine {  
  
    State soldOutState;  
    State noCoinState;  
    State hasCoinState;  
    State soldState;  
  
    State state = soldOutState;  
    int count = 0;  
  
    public GumballMachine(int numberGumballs)  
    {  
        soldOutState = new SoldOutState(this);  
        noCoinState = new NoQuarterState(this);  
        hasCoinState = new HasCoinState(this);  
        soldState = new SoldState(this);  
  
        this.count = numberGumballs;  
        if (numberGumballs > 0) {  
            state = noCoinState;  
        }  
    }  
}
```

```
        public void insertCoin() {  
  
    void setState(State state) {  
        this.state = state;  
    }  
  
    public State getState() {  
        return state;  
    }  
  
    public State getSoldOutState() {  
        return soldOutState;  
    }  
  
    public State getNoCoinState() {  
        return noCoinState;  
    }  
  
    public State getHasCoinState() {  
        return hasCoinState;  
    }  
  
    public State getSoldState() {  
        return soldState;  
    }  
}
```


Example

State Pattern

Change Request:



One in Ten get a FREE GUMBALL!

```
public class WinnerState implements State {  
    /* ... */  
}
```

```
public class GumballMachine {  
    /* ... */  
    State winnerState;  
    /* ... */  
}
```

Example

State Pattern

Change Request:



One in Ten get a FREE GUMBALL!

```
public class GumballMachine {  
    /* ... */  
}
```

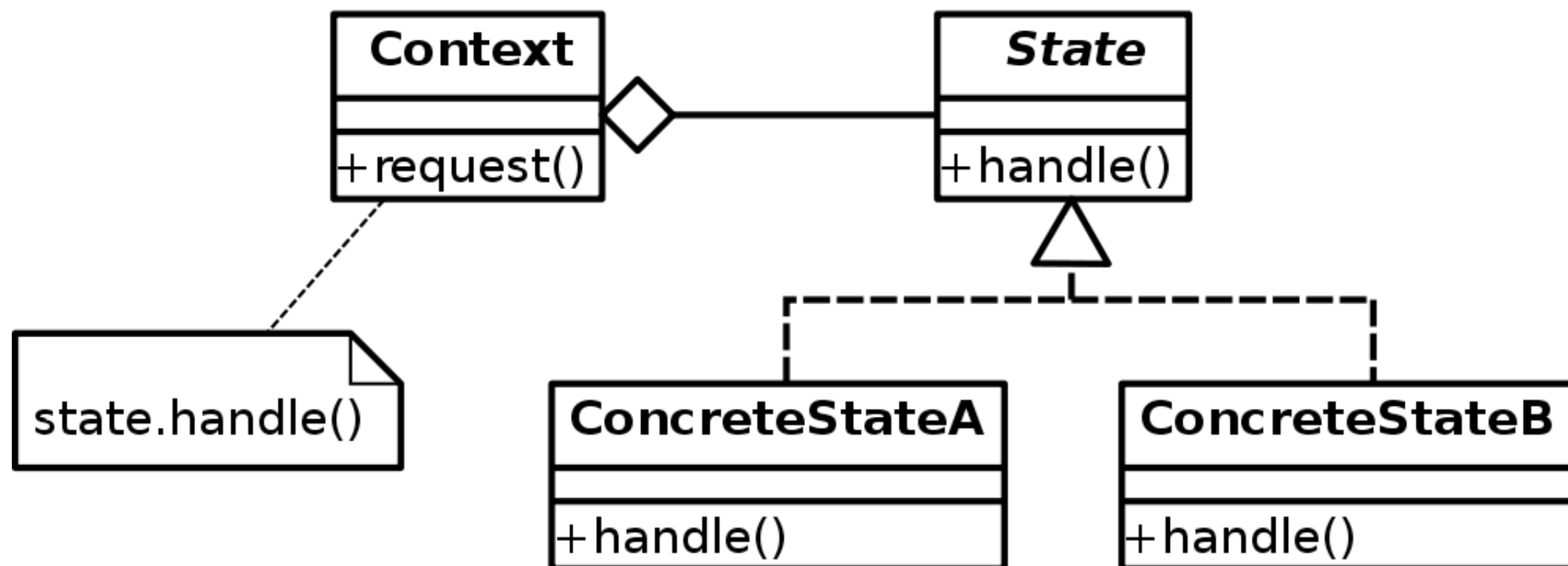
```
public class HasQuarterState implements State {  
    /* ... */  
}
```

```
public class HasQuarterState implements State {  
    Random randomWinner = new Random(System.currentTimeMillis());  
    /* ... */  
    public void turnCrank() {  
        System.out.println("You turned...");  
        int winner = randomWinner.nextInt(10);  
        if ((winner == 0) && (gumballMachine.getCount() > 1)) {  
            gumballMachine.setState(gumballMachine.getWinnerState());  
        } else {  
            gumballMachine.setState(gumballMachine.getSoldState());  
        }  
    }  
    /* ... */  
}
```

State Pattern

Design Patterns

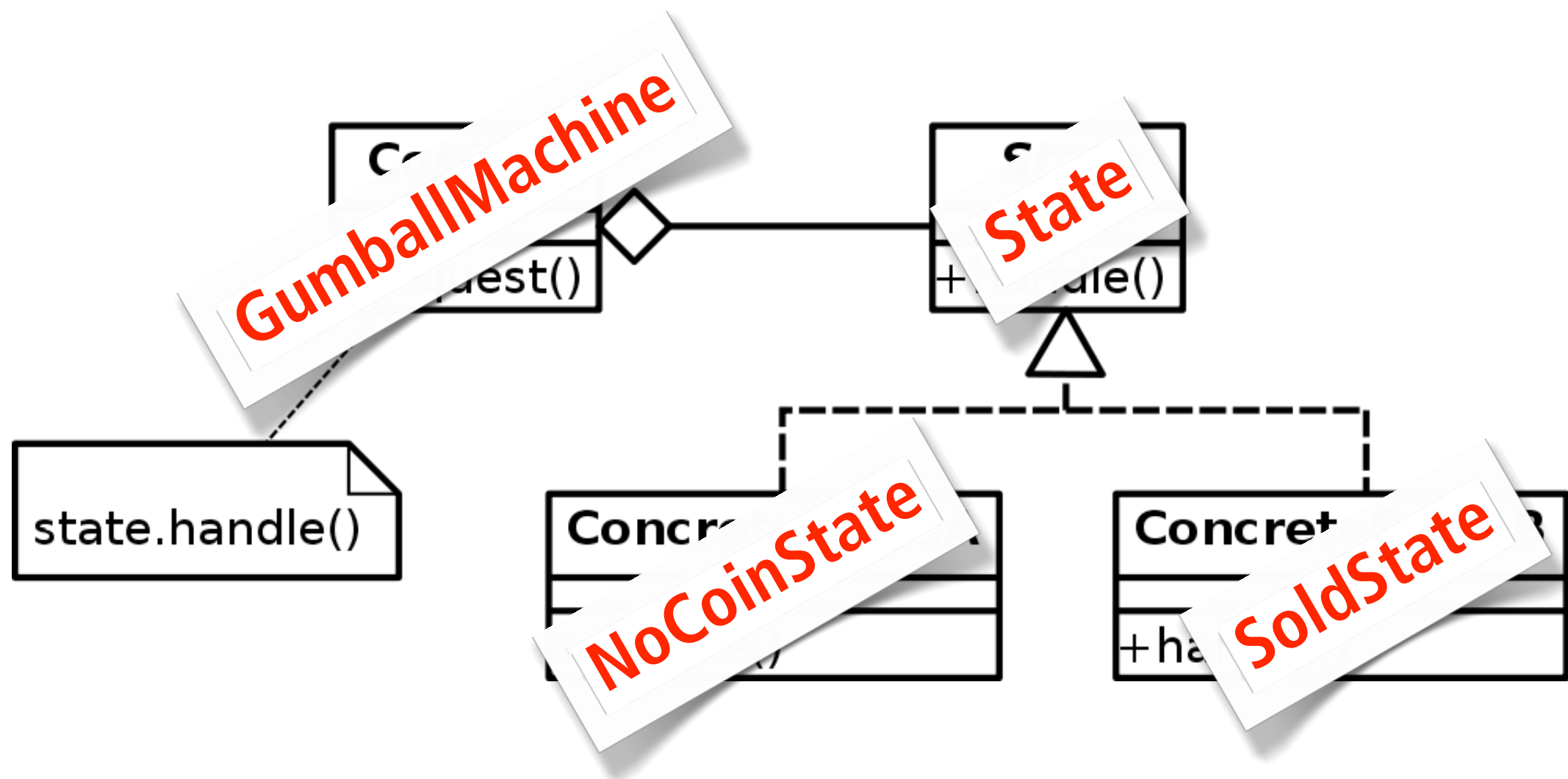
The **State Pattern** allows an object to alter its behavior when its internal state changes. The object will appear to change its class.



State Pattern

Design Patterns

The **State Pattern** allows an object to alter its behavior when its internal state changes. The object will appear to change its class.



Design Patterns

SoPra 2012

Head First Design Patterns by Eric T. Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra



Design Patterns - Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides



JUnit Testing SoPra 2012

- **test units in isolation:** concentrate on possible bugs within smallest possible units
- **regression tests:** develop a set of tests that can be run after each modification of the code
- **JUnit 4**
- later: **integration tests** with JUnit

Example JUnit Testing

```
public class PriceCalculator {  
    private Map<Integer, Double> taxTable = initTaxTable();  
  
    public double calculatePriceWithTax(double input, int year) {  
        return input * taxTable.get(year);  
    }  
  
    private Map<Integer, Double> initTaxTable() {  
        /* init tax table */  
    }  
}
```

Example JUnit Testing

```
public class PriceCalculator {  
    private Map<Integer, Double> taxTable = initTaxTable();  
  
    public double calculatePriceWithTax(double input, int year) {  
        return input * taxTable.get(year);  
    }  
  
    private Map<Integer, Double> initTaxTable() {  
        /* init tax table */  
    }  
}
```

```
public class PriceCalculatorTest {  
  
    @Test  
    public void calculatePriceWithTax() {  
        PriceCalculator calculator = new PriceCalculator();  
        double priceWithTax = calculator.calculatePriceWithTax(100.0, 2011);  
        assertEquals(108.00, priceWithTax, 0.0);  
    }  
}
```


Example JUnit Testing

```
public class PriceCalculator {  
    private Map<Integer, Double> taxTable = initTaxTable();  
  
    public double calculatePriceWithTax(double input, int year) {  
        return input * taxTable.get(year);  
    }  
  
    private Map<Integer, Double> initTaxTable() {  
        /* init tax table */  
    }  
}
```

```
public class PriceCalculatorTest {  
    @Test  
    public void calculatePriceWithTax() {  
        PriceCalculator calculator = new PriceCalculator();  
        double priceWithTax = calculator.calculatePriceWithTax(100.0, 2011);  
        assertEquals(108.00, priceWithTax, 0.0);  
    }  
}
```

Assert

JUnit Testing

- assertEquals
- assertEquals
- assertFalse
- assertNotNull
- assertNotSame
- assertNull
- assertSame
- assertThat
- assertTrue

Example

JUnit Testing

```
public class PriceCalculator {  
    private Map<Integer, Double> taxTable = initTaxTable();  
  
    public double calculatePriceWithTax(double input, int year) {  
        return input * taxTable.get(year);  
    }  
  
    private Map<Integer, Double> initTaxTable() {  
        /* init tax table */  
    }  
}
```

@RunWith(Parameterized.class)

public class PriceCalculatorTest {

@Parameters

public static Collection data() {

return Arrays.asList(new Object[][] {
/* Price Year PriceWithTax */
{ 100.0, 2010, 108.0 },
{ 0.0, 2010, 0.0 },
/* ..., ..., ... */
});

private double price;

private int year;

private double expectedPriceWithTax;

public PriceCalculatorTest(double price, int year, double expectedPriceWithTax) {

this.price = price;

this.year = year;

this.expectedPriceWithTax = expectedPriceWithTax;

}

@Test

public void calculatePriceWithTax() {

PriceCalculator calculator = new PriceCalculator();

double priceWithTax = calculator.calculatePriceWithTax(price, year);

assertEquals(expectedPriceWithTax, priceWithTax, 0.0);

}

}

JUnit 4

JUnit Testing

- **@BeforeClass**: called once before any test
- **@AfterClass**: called once after all tests
- **@Before**: called before each test
- **@After**: called after each test
- **@Test**: define a method as a test
- **@Parameters**: define test data

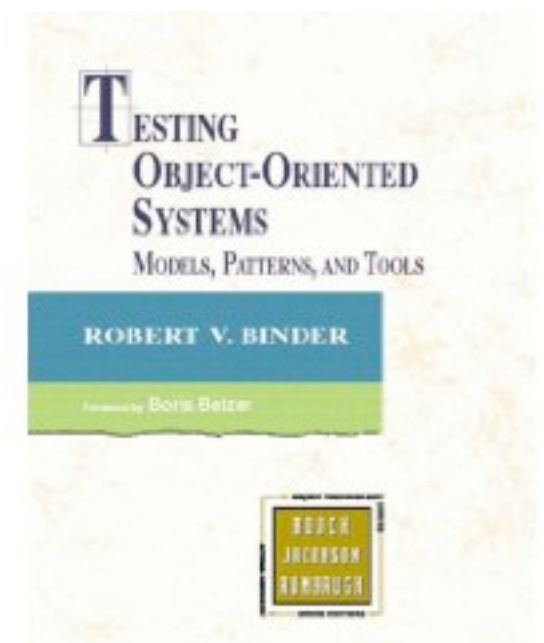
JUnit 4

JUnit Testing

Java Power Tools by John Ferguson Smart



**Testing Object-Oriented Systems:
Models, Patterns, and Tools** by Rober Binder



Hint!

SoPra 2012

- **Simple RMI Example with State Pattern**
- **RMI Example with State Pattern and Notification (Observer Pattern)**

<http://seal.ifi.uzh.ch/sopra> ➡ RMI State Pattern Examples

can be used as basis for your game implementation!

Next Review_{09.03.12}

- **Origo Account** wurde erstellt
- **Kommunikation** ist organisiert
- Alle **Use Cases** wurden erstellt
- Alle Elemente des Spiels wurden in einem **Domain Model** umgesetzt
- **Java RMI** Tutorial wurde durchgearbeitet
- Merkblatt zur Projektorganisation durchgelesen