CSCI 253

Object Oriented Design:
Composite Pattern
George Blankenship

Overview

Creational Patterns
- Singleton
- Abstract factory
- Factory Method
- Prototype
- Builder

Structural Patterns
- Composite
- Façade
- Proxy
- Flyweight
- Adapter
- Bridge
- Decorator

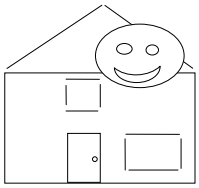
Behavioral Patterns
- Chain of Respons.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Composite Pattern: The Problem

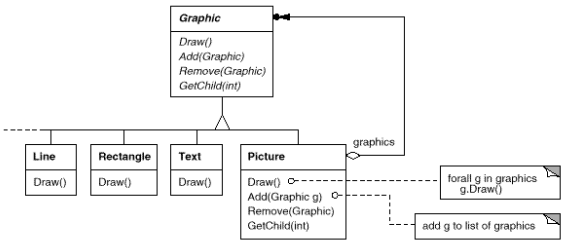
Compose objects into tree-like structures to represent part-whole hierarchies and let clients treat individual objects and compositions of objects uniformly



- a drawing tool that lets users build complex diagrams from simple elements
- trees with heterogeneous nodes e.g. the parse tree of a program
- a containment hierarchy for technical equipment

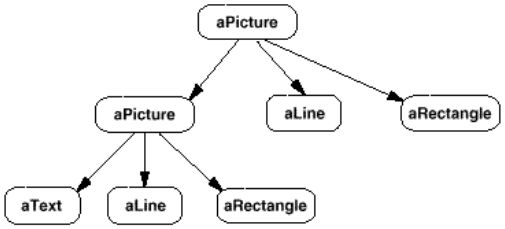
Composite Pattern George Blankenship 4

Graphic System Classes



Composite Pattern George Blankenship 5

Graphic System Objects



Composite Pattern George Blankenship 6

StateMachine

- public synchronized void setActive(boolean b) - set active
- public boolean getActive() {return active;} - get active state
- public String getName() {return name;} - get name of the machine
- public synchronized int getNextIndex() - get next state variable index
- public void close() - close machine
- public synchronized void setCurrentState(StateVariable s, long ms)
- public synchronized void goNextState(long ms)
- public void completed() - process ending states (normal/error)
- public boolean prologue() - prologue before processing state
- public boolean epilogue(StateMachine fsm) - epilogue after processing state
- public boolean execute() - execute state machine (state & input)
- public void timer() - execute state machine (timer)
- public void add(StateVariable s) - add SV to SV tree
- public String getShortSummary() - create summary of activity
- public void summarize() - summarize state machine

Composite Pattern George Blankenship 7

StateMachine States

- currentState =
  - new StateVariable(getName()+" current state");
- initialState =
  - new StateVariable(getName()+" initial state");
- endState =
  - new StateVariable(getName()+" end state");
- errorState =
  - new StateVariable(getName()+" error state");

Composite Pattern George Blankenship 8

EchoConnection States

- waitDSRState =
  - new StateVariable(CODE_FILE+" waiting for DSR");
- waitCTSState =
- new StateVariable(CODE_FILE+" waiting for CTS",10000);
- waitXONState =
  - new StateVariable(CODE_FILE+" waiting for XON",10000);
- waitInputState =
  - new StateVariable(CODE_FILE+" waiting for input");

Composite Pattern George Blankenship 9

DialConnection States

- waitDSRState =
  - new StateVariable(CODE_FILE+" waiting for DSR",10000);
- waitCTSState =
  - new StateVariable(CODE_FILE+" waiting for CTS",10000);
- waitXONState =
  - new StateVariable(CODE_FILE+" waiting for XON",10000);
- resetStringSendState =
  - new StateVariable(CODE_FILE+" send modem reset",1);
- initStringSendState =
  - new StateVariable(CODE_FILE+" send modem init",1);
- dialStringSendState =
  - new StateVariable(CODE_FILE+" send modem dial",1);
- resetStringSentState =
  - new StateVariable(CODE_FILE+" sent modem reset",initStringSendState,1000);
- initStringSentState =
  - new StateVariable(CODE_FILE+" sent modem init",initStringSendState,1000);
- dialStringSentState =
  - new StateVariable(CODE_FILE+" sent modem dial",dialStringSendState,60000);

Composite Pattern George Blankenship 10

AnswerConnection States

- waitDSRState =
  - new StateVariable(CODE_FILE+" waiting for DSR",10000);
- waitCTSState =
  - new StateVariable(CODE_FILE+" waiting for CTS",10000);
- waitXONState =
  - new StateVariable(CODE_FILE+" waiting for XON",10000);
- waitCDState =
  - new StateVariable(CODE_FILE+" waiting for CD");
- waitInputState =
  - new StateVariable(CODE_FILE+" waiting for input");
- resetStringSendState =
  - new StateVariable(CODE_FILE+" send modem reset",1);
- initStringSendState =
  - new StateVariable(CODE_FILE+" send modem init",1);
- endStringSendState =
  - new StateVariable(CODE_FILE+" send modem reset (close)",1);
- resetStringSentState =
  - new StateVariable(CODE_FILE+" sent modem reset",initStringSendState,1000);
- initStringSentState =
  - new StateVariable(CODE_FILE+" sent modem init",initStringSendState,1000);
- endStringSentState =
  - new StateVariable(CODE_FILE+" sent modem reset (close)",endStringSendState,60000);

Composite Pattern George Blankenship 11

SerialConnection Fragment

```
• } else if(name=="Echo") {                    // Echo command from connection menu
        trace.write("echo data received on port "+parameters.getPortName());
        if(parameters.isDialCircuit()==false) {
                if(parameters.isOpen()==false) {
                        FSM = new EchoConnection(mainGUI,connection,parameters);
                } else mainGUI.append(parameters.getPortName()+" already open");
        } else mainGUI.append(parameters.getPortName()+" is a dial circuit");
• } else if(name=="Dial") {                    // Dial command from connection menu
        trace.write("dial a phone number for port "+parameters.getPortName());
        if(parameters.isDialCircuit()) {
                if(parameters.isOpen()==false) {
                        FSM = new DialConnection(mainGUI,connection,parameters);
                } else mainGUI.append(parameters.getPortName()+" already open");
        } else  mainGUI.append(parameters.getPortName()+" not a dial circuit");
• } else if(name=="Answer") {                  // Dial command from connection menu
        trace.write("wait for an incoming call on port "+parameters.getPortName());
        if(parameters.isDialCircuit()) {
                if(parameters.isOpen()==false) {
                        FSM = new AnswerConnection(mainGUI,connection,parameters);
                } else mainGUI.append(parameters.getPortName()+" already open");
        } else mainGUI.append(parameters.getPortName()+" not a dial circuit");
```

Composite Pattern George Blankenship 12

SerialLink

- public static void main(String args[]) {

```
…
mainGUI = new GUI(Constants.PROGRAM,Constants.AUTHOR,configurationPanel);
…
connection = new SerialConnection(mainGUI,parameters);
…
mainGUI.append(Constants.PROGRAM+" ready!");
mainGUI.setClock((long) 0);
timerThread = new SerialLink();
timerThread.run();
```

- public void run() {

```
        mainGUI.append("timer running");
        while(GUI.stop()==false) {
                try {
                        Thread.sleep(sleepTime);
                } catch (InterruptedException e) {
                        break;
                }
                clock += sleepPeriod;
                mainGUI.setClock(clock);
                calendar = Calendar.getInstance();
                startTime = calendar.getTimeInMillis();
                FSM = (StateMachine) FSMs.getFirst();
                while(FSM!=null) {
                        FSM.timer();
                        FSM = (StateMachine) FSM.getNext();
                }
…
```

Composite Pattern George Blankenship 13

FSMvendorDataFlow

- waitConnectionState =
  - new StateVariable(getTrace(),this,getName()+" waiting for connection");
- sendSignUpState =
  - new StateVariable(getTrace(),this,getName()+" send StartUp",START_UP_WAIT);
- waitSignUp1ACKState =
  - new StateVariable(getTrace(),this,getName()+" wait SignUp ACK or QBP",ACK_WAIT);
- waitSignUp2ACKState =
  - new StateVariable(getTrace(),this,getName()+" wait SignUp ACK",ACK_WAIT);
- waitQBPState =
  - new StateVariable(getTrace(),this,getName()+" wait QBP (SignUp ACK'd)",QBP_WAIT);
- sendRSP1State =
  - new StateVariable(getTrace(),this,getName()+" send RSP (no SignUp ACK)",ACK_WAIT);
- sendRSP2State =
  - new StateVariable(getTrace(),this,getName()+" send RSP (SignUp ACK'd)",ACK_WAIT);
- waitRSP1ACKState =
  - new StateVariable(getTrace(),this,getName()+" wait for RSP ACK (no SignUp ACK)",ACK_WAIT);
- waitRSP2ACKState =
  - new StateVariable(getTrace(),this,getName()+" wait for RSP ACK (SignUp ACK'd)",ACK_WAIT);
- waitObservationState =
  - new StateVariable(getTrace(),this,getName()+" wait for ORU",OBSERVATION_WAIT);

Composite Pattern George Blankenship 14

PatientData

- private ICN icn;
- private DFN dfn;
- private SSN ssn;
- private XPNData patientName;
- private XPNData motherName;
- private ChameleonDateTime DOB;
- private String sex;
- private CEData race;
- private XADData address;
- private String homePhone;
- private String businessPhone;
- private String maritalStatus;
- private String religion;
- private String ethnicGroup;
- private String militaryStatus;
- private ChameleonDateTime DOD;
- private String deathIndicator;
- private XCNData provider;
- private Consult consult;

Composite Pattern George Blankenship 15

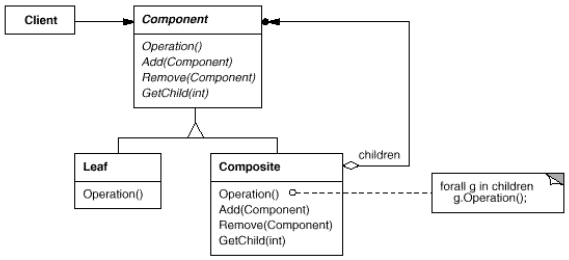
CXData

- private String identifier; // (of the data type)
- private Double checkDigit;
- private String checkDigitCode;
- private HDData authority;
- private String type;
- private HDData facility;
- private ChameleonDateTime effectiveDate;
- private ChameleonDateTime expirationDate;

Medical Record Numbers

- ICN
  - public ICN(String identifier)
  - public boolean isValid()
  - public String getKey()
- DFN
  - public DFN(String identifier, String facility, String station)
  - public boolean isValid()
  - public String getKey()
- SSN
  - public SSN(String identifier)
  - public boolean isValid()
  - public String getKey()

The Composite Pattern: Structure

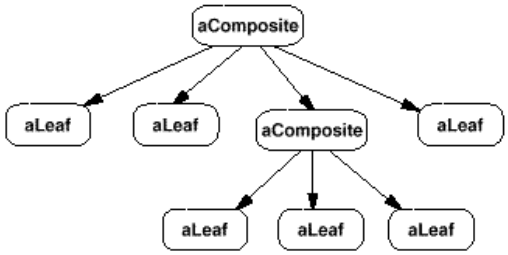


The Composite Pattern: Participants

- Component: declares the interface for objects in the composition, implements default behavior for the interface common to all objects, declares an interface for accessing and managing child components, (optional) defines/implements an interface for accessing a component's parent
- Leaf: defines behavior for primitive objects in the composition
- Composite: defines behavior for components having children, stores child components, implements child access and management operations in the component interface
- Client: manipulates objects in the composition through the component interface

Composite Pattern George Blankenship 19

The Composite Pattern: Object Structure



Composite Pattern George Blankenship 20

The Composite Pattern: Collaboration

- Clients use the Component class interface to interact with objects in the composition
- If the recipient is a Leaf, the request is handled directly
- If the recipient is a Composite the request is usually forwarded to child components, some additional operations before and/or after the forwarding can happen

Composite Pattern George Blankenship 21

The Composite Pattern: Consequences

- + Makes the Client simple: clients can treat composite structures and individual objects uniformly, clients normally don't know and should not care whether they are dealing with a leaf or a composite
- + Makes it easier to add new types of components: client code works automatically with newly defined Composite or Leaf subclasses
- - Can make a design overly general: the disadvantage of making it easy to add new components is that it is difficult to restrict the components of a composite, sometimes you want a composite to have only certain types of children, with the Composite Patterns you cannot rely on the type system to enforce this for you, you have to implement and use run-time checks

Composite Pattern George Blankenship 22

The Composite Pattern: Implementation

- Explicit parent references
- Sharing components
- Maximizing the Component interface
- The child access and management operations
- The instance variables hold the children
- Deleting components (non-Java)
- Data structure for storing children
- Child ordering
- Caching

Composite Pattern George Blankenship 23